Introduction to Algebraic Coding Theory

Supplementary material for Math 336 Cornell University

Sarah A. Spence

Contents

1 Introduction

Imagine that you are using an infrared link to beam messages consisting of 0s and 1s from your laptop to your friend's PalmPilot. Usually, when you send a 0, your friend's PalmPilot receives a 0. Occasionally, however, *noise* on the channel causes your 0 to be received as a 1. Examples of possible causes of noise include atmospheric disturbances. You would like to find a way to transmit your messages in such a way that errors are detected and corrected. This is where *error-control codes* come into play.

Error-control codes are used to detect and correct errors that occur when data is transmitted across some noisy channel. Compact discs (CDs) use error-control codes so that a CD player can read data from a CD even if it has been corrupted by noise in the form of imperfections on the CD. When photographs are transmitted to Earth from deep space, error-control codes are used to guard against the noise caused by lightning and other atmospheric interruptions.

Error-control codes build redundancy into a message. For example, if your message is $\mathbf{x} = 0$, you might encode $\mathbf{x}$ as the *codeword* $\mathbf{c} = 00000$. (We work more with this example in Chapter 2.) In general, if a message has length k, the encoded message, *i.e.* codeword, will have length $n > k$.

Algebraic coding theory is an area of discrete applied mathematics that is concerned (in part) with developing error-control codes and encoding/decoding procedures. Many areas of mathematics are used in coding theory, and we focus on the interplay between algebra and coding theory. The topics in this packet were chosen for their importance to developing the major concepts of coding theory and also for their relevance to a course in abstract algebra. We aimed to explain coding theory concepts in a way that builds on the algebra learned in Math 336. We recommend looking at any of the books in the bibliography for a more detailed treatment of coding theory.

As you read this packet, you will notice questions interspersed with the text. These questions are meant to be straight-forward checks of your understanding. You should work out each of these problems as you read the packet. More homework problems are found at the end of each chapter.

2 Basics

To get the ball rolling, we begin with some examples of codes. Our first example is an error-detecting, as opposed to error-correcting, code.

Example 2.1. (*The ISBN Code*) The International Standard Book Number (ISBN) Code is used throughout the world by publishers to identify properties of each book. The first nine digits of each ISBN represent information about the book including its language, publisher, and title. In order to guard against errors, the nine-digit "message" is encoded as a ten-digit codeword. The appended tenth digit is a *check digit* chosen so that the whole ten-digit string $x_1x_2\cdots x_{10}$ satisfies

$$\sum_{i=1}^{10} ix_i \equiv 0 \pmod{11}. \tag{1}$$

If x_{10} should be equal to 10, an 'X' is used.

The ISBN code can be used to detect any single error and any double-error created by the transposition of two digits. This detection scheme uses properties of $\mathbb{F}_{11}$, the finite field with 11 elements, as follows. Suppose we receive a length ten vector $y_1y_2\cdots y_{10}$ (by scanning a book). We calculate its weighted *check sum* $Y = \sum_{i=1}^{10} iy_i$. If $Y \neq 0$ modulo 11 , then we know that one or more errors has occurred. To prove that any single error can be detected, suppose

that $\mathbf{x} = x_1x_2\cdots x_{10}$ is sent, but $\mathbf{y} = y_1y_2\cdots y_{10}$ is received where $y_i = x_i$ for all $1 \leq i \leq 10$ except one index j where $y_j = x_j + a$ for some nonzero a. Then $Y = \sum_{i=1}^{10} iy_i = (\sum_{i=1}^{10} ix_i) + ja = ja \neq 0 \pmod{11}$, since j and a are nonzero. Since $Y \neq 0$, the single error is detected. To prove that any double-error created by the transposition of two digits is detected, suppose the received vector $\mathbf{y}$ is the same as the sent vector $\mathbf{x}$ except that digits x_j and x_k have been transposed. Then $Y = \sum_{i=1}^{10} iy_i = (\sum_{i=1}^{10} ix_i) + (k-j)x_j + (j-k)x_k = (k-j)(x_j - x_k) \neq 0 \pmod{11}$, if $k \neq j$ and $x_j \neq x_k$. *Notice that this last step relies on the fact that the field* $\mathbb{F}_{11}$ *has no zero divisors.*

Question 2.2. Are either of the following strings valid ISBNs: 0-13165332-6, 0-1392-4101-4?

The Universal Product Code (UPC) found on groceries and other items is another example of a code that employs a check digit. You can learn more about the UPC code at

http://www.howstuffworks.com/upc1.htm.

In some states, drivers' license numbers include check digits in order to detect errors or fraud. In fact, many states generate license numbers through the use of complicated formulas that involve both check digits and numbers based on information such as the driver's name, date of birth, and sex. Most states keep their formulas confidential, however Professor Joseph Gallian at the University of Minnesota-Duluth figured out how several states generate their license numbers. The following website summarizes some of his results and techniques:

http://www.maa.org/mathland/mathtrek_10_19_98.html.

Another website that has useful discussions about check digits is

http://www.cs.queensu.ca/home/bradbury/checkdigit/index.html.

Example 2.3. (*The Repetition Code*) This example is a simple error-correcting code. Suppose we would like to send a 1 to signify "yes", and a 0 to signify "no." If we simply send one bit, then there is a chance that noise will corrupt that bit and an unintended message will be received. A simple alternative is to use a *repetition code*. Instead of sending a single bit, we send 11111 to represent 1 (or yes), and 00000 to represent 0 (or no). Since the codewords consist of 0s and 1s, this is called a *binary* code. Alternatively, we can say that this code is defined over the finite field with two elements, $\mathbb{F}_2$. The receiver decodes a received 5-tuple as the bit that occurs most. This way, if zero, one or two errors occur, we will still decode the received 5-tuple as the intended message. In other words, the receiver decodes a received 5-tuple as the "closest" codeword. This is an example of *nearest neighbor decoding*, which is discussed in more detail in Section 2.2.

Notice that transmitting a message that has been encoded using this repetition codes takes five times longer than transmitting the uncoded message.

However, we have increased the probability of decoding the received string as the correct message.

Question 2.4. How many different binary 5-tuples exist? Which of these would be decoded as 00000?

2.1 Important code parameters

When developing codes, there must be a way to decide which codes are "good" codes. There are three main parameters used to describe and evaluate codes. The first parameter is the *code length*, n. In the repetition code of Example 2.3, the code length is 5, since the codewords 00000 and 11111 each contain 5 bits. We restrict our discussion to codes whose codewords are all of the same length. The next parameter that we consider is the *total number of codewords*, M. In Example 2.3, the total number of codewords is 2. The third parameter measures the *distance* between pairs of codewords in a code. In order to explain clearly the notion of distance between codewords, we need the following definitions.

Definition 2.5. The *Hamming weight* $w(\mathbf{c})$ of a codeword $\mathbf{c}$ is the number of nonzero components in the codeword.

Example 2.6. $w(00000) = 0$, $w(11111) = 5$, $w(1022001) = 4$, $w(1011001) = 4$.

Definition 2.7. The *Hamming distance* between two codewords $d(\mathbf{x}, \mathbf{y})$ is the Hamming weight of the vector difference $\mathbf{x} - \mathbf{y}$, i.e $w(\mathbf{x} - \mathbf{y})$.

That is, the Hamming distance between two codewords $d(\mathbf{x}, \mathbf{y})$ is the number of places in which $\mathbf{x}$ and $\mathbf{y}$ differ. There are other distances used to evaluate codes, but in this packet, distance will always refer to Hamming distance. To read about other distance functions used to evaluate codes, see [7].

Question 2.8. Calculate $d(00111, 11001)$. How about $d(00122, 12001)$?

Definition 2.9. The *minimum (Hamming) distance* of a code C is the minimum distance between any two codewords in the code: $d(C) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \neq \mathbf{y},\ \mathbf{x}, \mathbf{y} \in C\}$.

In Example 2.3, the minimum distance is 5 since the two codewords differ in all 5 positions.

Question 2.10. What is the minimum distance between any two ISBN codewords?

The Hamming distance is a *metric* on the space of all n-tuples over $\mathbb{F}_q$ which means that d satisfies the following properties for any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$:

1. $d(\mathbf{x}, \mathbf{x}) = 0$
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
3. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$

The third condition above is called the *triangle inequality*. The proofs of the above properties are left as problems at the end of this chapter.

The following notation is standard and will be used throughout this packet. An (n, M, d) code has minimum distance d and consists of M codewords, all of length n. One of the major goals of coding theory is to develop codes that strike a balance between having small n (for fast transmission of messages), large M (to enable transmission of a wide variety of messages), and large d (to detect many errors).

Traditionally, the alphabets used in coding theory are finite fields, $\mathbb{F}_q$. We say that a code is q-ary if its codewords are defined over the q-ary alphabet $\mathbb{F}_q$. The most commonly used alphabets are binary extension fields, $\mathbb{F}_{2^m}$. In the common case where the alphabet is $\mathbb{F}_2$, we say that the code is *binary*.

Example 2.11. Let $C = \{0000, 1100, 0011, 1111\}$. Then C is a $(4, 4, 2)$ binary code.

2.2 Correcting and detecting errors

When we talk about the number of errors in a received codeword, we are talking about the distance between a received word and a transmitted word. Suppose a codeword $\mathbf{x} = x_0x_1 \dots x_{n-1}$ is sent through a channel and the received vector is $\mathbf{y} = y_0y_1 \dots y_{n-1}$. The error vector is defined as $\mathbf{e} = \mathbf{y} - \mathbf{x} = e_0e_1 \dots e_{n-1}$. The job of the decoder is to decide which codeword was most likely transmitted, or equivalently, decide which error vector most likely occurred. Many codes use a *nearest neighbor decoding scheme* which chooses the codeword that minimizes the distance between the received vector and possible transmitted vectors. A nearest neighbor decoding scheme for a q-ary code maximizes the decoder's likelihood of correcting errors provided the following assumptions are made about the channel.

1. Each symbol transmitted has the same probability p ($< 1/2$) of being received in error.

2. If a symbol is received in error, that each of the $q - 1$ possible errors is equally likely.

Such a channel is called a q-ary *symmetric channel*, and we assume throughout this packet that the channels involved are symmetric.

If a binary symmetric channel is assumed and if a particular binary codeword of length n is transmitted, then the probability that no errors will occur is $(1 - p)^n$, since each symbol has probability $(1 - p)$ of being received correctly.

Let's revisit Example 2.3. Recall that if zero, one, or two errors occur while using the repetition code of length 5, then we still decode correctly. If we are using a binary symmetric channel, then the probability of decoding a word correctly is

$$(1 - p)^5 + 5p(1 - p)^4 + 10p^2(1 - p)^3 \tag{2}$$

Question 2.12. Explain how the expression in (2) was obtained.

We know that the repetition code of length 5 corrects up to two errors, and it clearly can detect up to four errors. In general, a code that has minimum distance d can be used to either detect up to $d-1$ errors or correct up to $\lfloor (d-1)/2 \rfloor$ errors. This is a consequence of the following theorem.

Theorem 2.13. 1. A code C can detect up to s errors in any codeword if $d(C) \geq s+1$.

2. A code C can correct up to t errors in any codeword if $d(C) \geq 2t+1$.

Proof. 1. Suppose $d(C) \geq s+1$. Suppose a codeword $\mathbf{c}$ is transmitted and that s or fewer errors occur during the transmission. Then, the received word can not be a different codeword, since all codewords differ from $\mathbf{c}$ in at least $s+1$ places. Hence, the errors are detected.

2. Suppose $d(C) \geq 2t+1$. Suppose a codeword $\mathbf{x}$ is transmitted and that the received word, $\mathbf{y}$, contains t or fewer errors. Then $d(\mathbf{x}, \mathbf{y}) \leq t$. Let $\mathbf{x}'$ be any codeword other than $\mathbf{x}$. Then $d(\mathbf{x}', \mathbf{y}) \geq t+1$, since otherwise $d(\mathbf{x}', \mathbf{y}) \leq t$ which implies that $d(\mathbf{x}, \mathbf{x}') \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{x}', \mathbf{y}) \leq 2t$ (by the triangle inequality), which is impossible since $d(C) \geq 2t+1$. So $\mathbf{x}$ is the nearest codeword to $\mathbf{y}$, and $\mathbf{y}$ is decoded correctly.

□

Example 2.14. (*Transmission of photographs from deep-space*) This example describes a method for transmitting photographs. It is the method that was used in the Mariner 1969 Mission which sent photographs of Mars back to Earth. In order to transmit a picture, the picture is first divided into very small squares, known as pixels. Each pixel is assigned a number representing its degree of blackness, say in a scale of 0 to 63. These numbers are expressed in the binary system, where 000000 represents white and 111111 represents black. Each binary 6-tuple is encoded using a (32, 64, 16) code, called a Reed-Muller code. Reed-Muller codes are often used in practice because they are very easy to decode.

Question 2.15. How many errors can the (32, 64, 16) Reed-Muller code correct?

Suppose that instead of the Reed-Muller code, we use a repetition code of length 5 to transmit each bit of each pixel's color assignment. Then each 6-bit message string (representing a color assignment) is encoded as a codeword of length 30, since each bit in the message string is repeated five times. How can we compare performance of this usage of a repetition code with the performance of the (32, 64, 16) Reed-Muller code in Example 2.14? One quantity used to compare codes is the *rate* of a code.

Definition 2.16. The *rate* of a code is the ratio of message bits to coded bits.

Since each codeword of length 30 in the repetition code represents a 6-bit message, the rate of the code is 1/5. Since each codeword of length 32 in the

Reed-Muller code in Example 2.14 represents a 6-bit message, the rate of the code is 6/32. Lower rate codes are faster and use less power, however sometimes higher rate codes are preferred because of superior error-correcting capability or other practical considerations.

We'd like to compare the probability of error when using these two codes. First, we need some notation. The number $\binom{n}{m}$, read "n choose m," counts the number of ways that we can choose m objects from a pool of n objects, and $\binom{n}{m} = n!/(n-m)!m!$. Note that $n!$ is read "n factorial," and $n! = n \cdot (n-1) \cdots 2 \cdot 1$. The numbers of the form $\binom{n}{m}$ are called *binomial coefficients* because of the binomial theorem which states that for any positive integer n,

$$(1+x)^n = 1 + \binom{n}{1}x + \binom{n}{2}x^2 + \cdots + \binom{n}{n}x^n.$$

We now return to our code analysis. When using the Reed-Muller code, the probability that a length 32 codeword is decoded incorrectly is

$$\sum_{i=8}^{32} \binom{32}{i} p^i (1-p)^{32-i} \tag{3}$$

Question 2.17. Explain how the expression in (3) was obtained.

If we use a repetition code of length 5 to encode the 6-bit message strings, then the probability that one of the resulting codewords of length 30 is received correctly is equal to the expression in (2) raised to the 6th power:

$$[(1-p)^5 + 5p(1-p)^4 + 10p^2(1-p)^3]^6 \tag{4}$$

Question 2.18. Explain how the expression in (4) was obtained.

Question 2.19. Suppose that $p = .01$ and compare the probability of decoding incorrectly when using the repetition code with the probability of decoding incorrectly when using the Reed-Muller code.

Question 2.20. Using code rate and probability of decoding incorrectly to compare the repetition code and the Reed-Muller code, which code do you think is an overall better code?

2.3 Sphere-packing bound

We know from Theorem 2.13 that if C has minimum distance $d \geq 2t+1$, then C can correct at least t errors. We now develop a geometric interpretation of this result. Since each codeword in C is at least distance $2t+1$ from any other codeword, we can picture each codeword $\mathbf{x}$ to be surrounded by a sphere of radius t such that the spheres are disjoint. Any received vector that lies within the sphere centered at $\mathbf{x}$ will be decoded as $\mathbf{x}$. This pictorially explains why received vectors that contain t or fewer errors are decoded correctly: They still lie within the correct sphere. We can use this picture to bound the total possible number of codewords in a code, as seen in the following theorem:

Theorem 2.21. (*Sphere-packing bound*) A t-error-correcting q-ary code of length n must satisfy

$$M \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n \tag{5}$$

where M is the total number of codewords.

In order to prove Theorem 2.21, we need the following lemma.

Lemma 2.22. A sphere of radius r, $0 \leq r \leq n$, in $\mathbb{F}_q^n$ contains exactly

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

vectors.

Proof of Lemma 2.22. Let $\mathbf{u}$ be a fixed vector in $\mathbb{F}_q^n$. Consider how many vectors $\mathbf{v}$ have distance exactly m from $\mathbf{u}$, where $m \leq n$. The m positions in which $\mathbf{v}$ is to differ from $\mathbf{u}$ can be chosen in $\binom{n}{m}$ ways, and then in each of these m positions the entry of $\mathbf{v}$ can be chosen in $q-1$ ways to differ from the corresponding entry of $\mathbf{u}$. Hence, the number of vectors at distance exactly m from $\mathbf{u}$ is $\binom{n}{m}(q-1)^m$. So, the total number of vectors in a ball of radius r, centered at $\mathbf{u}$, must be $\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{r}(q-1)^r$. □

Proof of Theorem 2.21. Suppose C is a t-error-correcting q-ary code of length n. As explained in the discussion above Theorem 2.21, in order that C can correct t errors, any two spheres of radius t centered on distinct codewords can have no vectors in common. Hence, the total number of vectors in the M spheres of radius t centered on the M codewords of C is given by $M \sum_{i=0}^{t} \binom{n}{i}(q-1)^i$, by Lemma 2.22. This number of vectors must be less than or equal to the total number of vectors in $\mathbb{F}_q^n$, which is q^n. This proves the sphere-packing bound. □

Definition 2.23. A code is *perfect* if it satisfies the sphere-packing bound of Theorem 2.21 with equality.

When decoding using a perfect code, every possible word in $\mathbb{F}_q^n$ is at distance less than or equal to t from a unique codeword, so the nearest neighbor algorithm yields an answer for every received word $\mathbf{y}$, and it is the correct answer when the number of errors is less than or equal to t. In the next chapter, we will introduce a family of perfect codes called Hamming codes.

2.4 Problems

1. Complete the ISBN that starts as 0-7803-1025-.
2. Let $C = \{00, 01, 10, 11\}$. Why can't C correct any errors?
3. Prove that the Hamming distance satisfies the following conditions for any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$:

(a) $d(\mathbf{x}, \mathbf{x}) = 0$

(b) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$

(c) $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$

4. Let C be a code that contains $\mathbf{0}$, the codeword consisting of a string of all zeros. Suppose that C contains a string $\mathbf{u}$ as a codeword. Now suppose that the string $\mathbf{u}$ also occurs as an error. Prove that C will not always detect when the error $\mathbf{u}$ occurs.

5. Let C be a code of even minimum distance $d(C) = 2t$. Show that C can be used to simultaneously correct up to $t-1$ errors and to detect t errors. More precisely, give an algorithm that takes a received word $\mathbf{y}$ and either produces a code word $\mathbf{x}$ or declares "t errors;" the algorithm should always give the right answer if the number of errors is at most t. Explain why, in spite of Theorem 2.13, we can't hope to detect $2t-1$ errors if we are simultaneously trying to correct errors.

6. Prove that for $n = 2^r - 1$, $\binom{n}{0} + \binom{n}{1} = 2^r$.

7. Prove that any ternary $(11, 3^6, 5)$ code is perfect.

8. (a) Show that a 3-ary $(3, M, 2)$ code must have $M \leq 9$.

   (b) Show that a 3-ary $(3, 9, 2)$ code does exist.

   (c) Generalize the results from the above two parts to q-ary $(3, M, 2)$ codes, for any integer q.

9. Let E_n denote the set of all vectors in $\mathbb{F}_2^n$ which have even wight. Show that E_n is the code obtained by adding an overall parity check to the code consisting of all vectors in $\mathbb{F}_2^{n-1}$. Deduce that E_n is an $(n, 2^{n-1}, 2)$ code.

10. Show that a q-ary $(q+1, M, 3)$ code satisfies $M \leq q^{q-1}$.

11. Prove that all binary repetition codes of odd length are perfect codes.

12. Prove that no repetition code of even length can be perfect.

3 Linear codes

In this chapter, we study error-control codes that have additional structure.

Definition 3.1. A *linear code* of length n over $\mathbb{F}_q$ is a subspace of the vector space $\mathbb{F}_q^n$.

Hence, a subset C of $\mathbb{F}_q^n$ is a *linear code* if and only if (1) $\mathbf{u}+\mathbf{v} \in C$ for all $\mathbf{u}$, $\mathbf{v}$ in C, and (2) $a\mathbf{u} \in C$, for all $\mathbf{u} \in C$ and $a \in \mathbb{F}_q$. Linear codes are widely used in practice for a number of reasons. One reason is that they are easy to find. Another reason is that encoding linear codes is very quick and easy. Decoding is also often facilitated by the linearity of a code.

When we are viewing $\mathbb{F}_q^n$ as a vector space, we will write $V(n,q)$. If C is a k-dimensional subspace of $V(n,q)$, we say that C is an $[n,k,d]$ or $[n,k]$ linear code, and we can talk about the dimension k of the code. If C is a q-ary $[n,k]$ linear code, then C has q^k codewords. This means that C can be used to communicate any of q^k distinct messages. We identify these messages with the q^k elements of $V(k,q)$, the vector space of k-tuples over $\mathbb{F}_q$. The idea is to encode messages of length k to get codewords of length n, where $n > k$.

Linear codes have several useful properties, two of which are described in the following theorems.

Theorem 3.2. Let C be a linear code. The linear combination of any set of codewords in C is a code word in C.

Proof. C is a subspace of $V(n,q)$, and this property follows directly from the definition of a vector space. □

Question 3.3. Is $\{100, 001, 101\}$ a linear code?

Question 3.4. Show that the codeword $\mathbf{0}$ consisting of all zeros is always contained in a linear code.

Theorem 3.5. The minimum distance, $d(C)$, of a linear code C is equal to $w^*(C)$, the weight of the lowest-weight nonzero codeword.

Proof. There exist codewords $\mathbf{x}$ and $\mathbf{y}$ in C such that $d(C) = d(\mathbf{x}, \mathbf{y})$. By the definition of Hamming distance, we can rewrite this as $d(C) = w(\mathbf{x} - \mathbf{y})$. Since $\mathbf{x} - \mathbf{y}$ is a codeword in C by Theorem 3.2 and by the definition of $w^*(C)$, we have $d(C) = w(\mathbf{x} - \mathbf{y}) \geq w^*(C)$.

On the other hand, there exists some codeword $\mathbf{c} \in C$ such that $w^*(C) = w(\mathbf{c}) = d(\mathbf{c}, \mathbf{0}) \geq d(C)$, since $\mathbf{0} \in C$. Since we have now shown that both $d(C) \geq w^*(C)$ and $d(C) \leq w^*(C)$, we conclude that $d(C) = w^*(C)$. □

Theorem 3.5 greatly facilitates finding the minimum distance of a linear code. Instead of looking at the distances between all possible pairs of codewords, we need only look at the weight of each codeword.

Question 3.6. How many pairs of codewords would we need to consider if trying to find the minimum distance of a nonlinear code with M codewords?

Sometimes we can start with a known binary code and add an *overall parity check digit* to increase the minimum distance of a code. Suppose C is an (n, M, d) code. Then we can construct a code C', called the *extended code* of C, by adding a parity check digit to each codeword $\mathbf{x} \in C$ to obtain a codeword $\mathbf{x}' \in C'$ as follows. For each $\mathbf{x} = x_0 x_1 \cdots x_{n-1} \in C$, let $\mathbf{x}' = x_0 x_1 \cdots x_{n-1} 0$ if the Hamming weight of $\mathbf{x}$ is even, and let $\mathbf{x}' = x_0 x_1 \cdots x_{n-1} 1$ if the Hamming weight of $\mathbf{x}$ is odd. This ensures that every codeword in the extended code has even Hamming weight.

Theorem 3.7. Let C be a binary linear code with minimum distance $d = d(C)$. If d is odd, then the minimum distance of the extended code C' is $d + 1$, and if d is even, then the minimum distance of C' is d.

Proof. In the problems at the end of this chapter, you will prove that the extended code C' of a binary linear code is also a linear code. Hence by Theorem 3.5, $d(C')$ is equal to the smallest weight of nonzero codewords of C'. Compare the weights of codewords $\mathbf{x}' \in C'$ with the weights of the corresponding codewords $\mathbf{x} \in C$: $w(\mathbf{x}') = w(\mathbf{x})$ if $w(\mathbf{x})$ is even, and $w(\mathbf{x}') = w(\mathbf{x})+1$ if $w(\mathbf{x})$ is odd. By Theorem 3.5, we can conclude that $d(C') = d$ if d is even and $d(C') = d+1$ if d is odd. □

3.1 Generator and parity check matrices

Since linear codes are vector spaces, an $[n, k]$ linear code can be specified by giving a basis of k codewords.

Definition 3.8. A $k \times n$ matrix G whose rows form a basis for an $[n, k]$ linear code is called a *generator matrix* of the code.

In fact, a matrix G is a generator matrix for some linear code C if and only if the rows of G are linearly independent.

Question 3.9. Write down a generator matrix for the code $C = \{0000, 1111\}$.

A generator matrix G for a linear code C provides a compact way to describe all of the codewords in C. Furthermore, G can be used to encode messages. By performing the multiplication $\mathbf{u}G$, a generator matrix maps a length k message string $\mathbf{u}$ to a length n codeword string. The encoding function $\mathbf{u} \to \mathbf{u}G$ maps the vector space $V(k, q)$ onto a k-dimensional subspace (namely the code C) of the vector space of $V(n, q)$.

Example 3.10. The matrix $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ encodes (1, 1, 1) as (1, 1, 1, 1) and (1, 0, 0) as (1, 0, 0, 0).

Notice that $\mathbf{u}G$ is a linear combination of the rows of the generator matrix, so C is the rowspace of G.

Recall the following definition from Chapter 11E of the main text [1]:

Definition 3.11. A *permutation* of a set A is a function from A to itself that is both one-to-one and onto.

The following definition shows one way in which permutations are used in coding theory.

Definition 3.12. Two q-ary linear codes are called *equivalent* if one can be obtained from the other by a combination of operations of the following types:

1. Permutation of the positions of the codewords
2. Multiplication of the symbols appearing in a fixed position by a nonzero scalar (*i.e.* element of $\mathbb{F}_q$).

Theorem 3.13. Two $k \times n$ matrices generate equivalent $[n, k]$ linear codes over $\mathbb{F}_q$ if one matrix can be obtained from the other by a sequence of operations of the following types:

1. Permutation of the rows
2. Multiplication of a row by a nonzero scalar
3. Addition of a scalar multiple of one row to another
4. Permutation of the columns
5. Multiplication of any column by a nonzero scalar.

Proof. The first three types of operations preserve the linear independence of the rows of a generator matrix and simply replace one basis of the code by another basis of the same code. The last two types of operations convert a generator matrix for one code into a generator matrix for an equivalent code. □

Recall the following definitions from linear algebra.

Definition 3.14. The *inner product* $\mathbf{u} \cdot \mathbf{v}$ of vectors $\mathbf{u} = (u_0, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, \ldots, v_{n-1})$ in $V(n, q)$ is the *scalar* defined by $\mathbf{u} \cdot \mathbf{v} = u_0v_0 + u_1v_1 + \cdots + u_{n-1}v_{n-1}$.

Question 3.15. In $V(4, 3)$, what is the inner product of (2, 0, 1, 1) and (1, 2, 1, 0)? *Hint: Make sure that your answer is a scalar in the correct field.*

Definition 3.16. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal* if $\mathbf{u} \cdot \mathbf{v} = 0$.

Definition 3.17. Given a subspace S of some vector space $V(n, q)$, the space of all vectors orthogonal to S is called the *orthogonal complement* of S, and is denoted $S^{\perp}$.

For example, you may recall that the nullspace of a matrix is the orthogonal complement of the rowspace of that matrix.

The following definition shows how the above concepts from linear algebra are used in coding theory.

Definition 3.18. Given a linear $[n, k]$ code C, the *dual code* of C, denoted $C^{\perp}$, is the set of vectors of $V(n, q)$ which are orthogonal to every codeword in C, *i.e.*

$$C^{\perp} = \{\mathbf{v} \in V(n, q) \mid \mathbf{v} \cdot \mathbf{u} = 0, \ \ \forall\, \mathbf{u} \in C\}$$

Hence, the concepts of dual codes and orthogonal complements in $V(n, q)$ are the same. However, the reader should be careful not to think of a dual code as an orthogonal complement in the sense of vector spaces over the real numbers $\mathbb{R}$: In the case of finite fields, C and $C^{\perp}$ can have intersections larger than $\{0\}$. In fact, codes where $C = C^{\perp}$ are called *self-dual* and are well-studied. If C is an $[n, k]$ linear code, then $C^{\perp}$ is an $[n, n - k]$ linear code. Furthermore, if C has generator matrix G, then $C^{\perp}$ has an $(n - k) \times n$ generator matrix H that satisfies $GH^T = 0$. The generator matrix H for $C^{\perp}$ is also called a *parity check matrix* for C, as explained by the following theorem.

Theorem 3.19. Let C be a linear code, and let H be a generator matrix for $C^\perp$, the dual code of C. Then a vector $\mathbf{c}$ is a codeword in C if and only if $\mathbf{c}H^T = \mathbf{0}$, or equivalently, if and only if $H\mathbf{c}^T = \mathbf{0}$.

Proof. Let $\mathbf{c} \in C$. Then $\mathbf{c} \cdot \mathbf{h} = 0$ for all $\mathbf{h} \in C^\perp$ by the definition of dual codes. It follows that $\mathbf{c}H^T = \mathbf{0}$, since the rows of H form a basis for $C^\perp$.

Alternately, let $\mathbf{c}$ be a vector such that $\mathbf{c}H^T = 0$. Then $\mathbf{c} \cdot \mathbf{h} = 0$ for all $\mathbf{h}$ in the dual code $C^\perp$. So $\mathbf{c} \in (C^\perp)^\perp$. You will prove at the end of this Chapter that $(C^\perp)^\perp = C$, hence $\mathbf{c} \in C$. □

Theorem 3.19 motivates the following definition:

Definition 3.20. Let C be an $[n, k]$ cyclic code. A *parity check matrix* for C is an $(n-k) \times n$ matrix H such that $\mathbf{c} \in C$ if and only if $\mathbf{c}H^T = 0$. Equivalently, a parity check matrix for C is a generator matrix for $C^\perp$.

Theorem 3.19 shows that C is equal to the nullspace of the parity check matrix H. We can use the parity check matrix of a code to determine the minimum distance of the code:

Theorem 3.21. Let C have parity check matrix H. The minimum distance of C is equal to the minimum nonzero number of columns in H for which a nontrivial linear combination of the columns sums to zero.

Proof. Since H is a parity check matrix for C, $\mathbf{c} \in C$ if and only if $\mathbf{0} = \mathbf{c}H^T$. Let the column vectors of H be $\{\mathbf{d_0}, \mathbf{d_1}, \ldots, \mathbf{d_{n-1}}\}$. The matrix equation $\mathbf{0} = \mathbf{c}H^T$ can be reexpressed as follows:

$$\begin{aligned} \mathbf{0} &= \mathbf{c}H^T & (6)\\ &= (c_0, c_1, \ldots, c_{n-1})[\mathbf{d_0}\ \mathbf{d_1}\ \cdots\ \mathbf{d_{n-1}}]^T & (7)\\ &= c_0\mathbf{d_0} + c_1\mathbf{d_1} + \cdots + c_{n-1}\mathbf{d_{n-1}} & (8) \end{aligned}$$

This shows that $\mathbf{c}$ is a weight $d > 0$ codeword if and only if there is a nontrivial linear combination of d columns of H which equals zero. It now follows from Theorem 3.5 that the minimum distance of C is equal to the minimum nonzero number of columns in H for which a nontrivial linear combination of the columns sums to zero. □

Question 3.22. Suppose that $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ is a parity check matrix for a binary code C. What is the minimum distance of C?

Theorem 3.21 can be used to bound the possible minimum distance for a code, as explained in the following theorem.

Theorem 3.23. *(Singleton bound)* The minimum distance d for an $[n, k]$ linear code satisfies $d \leq n - k + 1$.

Proof. An $[n, k]$ linear code has a parity check matrix H containing $(n - k)$ linearly independent rows. By linear algebra facts, any such H also has exactly $(n-k)$ linearly independent columns. It follows that any collection of $(n-k+1)$ columns of H must be linearly dependent. The result now follows from Theorem 3.21. □

In Chapter 4.5, we will study some codes that achieve the Singleton bound with equality. Such codes are called *maximum distance separable*.

An $(n-k) \times n$ parity check matrix H for a code C in *standard form* if $H = (A \mid I_{n-k})$. Then the generating matrix in *standard form* is $G = (I_k \mid -A^T)$. (Note: Authors vary in their definitions of the standard form of generator and parity check matrices.) Any parity check matrix or generating matrix can be put in standard form, up to rearranging of columns, by performing row operations. This changes neither the nullspace nor the rowspace of the matrix, which is important since C is the nullspace of H and the rowspace of G.

When a data block is encoded using a generator matrix in standard form, the data block is embedded without modification in the first k coordinates of the resulting codeword. This facilitates decoding. When data is encoded in this fashion, it is called *systematic encoding*.

3.2 Coset and syndrome decoding

This section discusses a nearest neighbor decoding scheme that uses the fact that a linear code is a subgroup of the additive group of the vector space $V(n, q)$. In particular, this section shows how cosets (first encountered in Chapter 11B of the main text [1]) can be used to decode a linear code.

Definition 3.24. Suppose that C is an $[n, k]$ linear code over $\mathbb{F}_q$ and that $\mathbf{a}$ is any vector in $V(n, q)$. Then the set $\mathbf{a} + C = \{\mathbf{a} + \mathbf{x} \mid \mathbf{x} \in C\}$ is called a *coset* of C.

Theorem 3.25. Suppose C is an $[n, k]$ linear code over $\mathbb{F}_q$. Then every vector of $V(n, q)$ is in some coset of C, every coset contains exactly q^k vectors, and two cosets are either disjoint or equal.

Proof. See Chapter 11 of the main text [1]. □

Example 3.26. Let C be the binary $[4, 2]$ linear code with generator matrix $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$. Then $C = \{0000, 1011, 0101, 1110\}$, and the cosets of C are

$$\begin{aligned} 0000 + C &= \{0000, 1011, 0101, 1110\} \\ 1000 + C &= \{1000, 0011, 1101, 0110\} \\ 0100 + C &= \{0100, 1111, 0001, 1010\} \\ 0010 + C &= \{0010, 1001, 0111, 1100\}. \end{aligned}$$

Notice that coset $0001 + C$ is equal to coset $0100 + C$, and we have only listed this coset once.

The *coset leader* of a coset is chosen to be one of the vectors of minimum weight in the coset. In the example above, either 0100 or 0001 could be chosen as the coset leader for the corresponding coset.

The coset decoding algorithm for an $[n, k]$ linear code works as follows. We partition $\mathbb{F}_q^n$ into cosets of C. There are $q^n/q^k = q^{n-k}$ cosets, each of which contains q^k elements. For each coset, pick a coset leader. There may be an arbitrary choice involved at this step. Then, if $\mathbf{y}$ is received, find the coset that contains $\mathbf{y}$. This coset is of the form $\mathbf{e} + C$, and we guess that $\mathbf{y} - \mathbf{e}$ was sent. Note that coset leaders, defined to be of least weight, represent error vectors, and this decoding algorithm assumes that lowest weight error vectors are the most probable.

In order to carry out the above decoding algorithm, we make a *standard array* by listing the cosets of C in rows, where the first entry in each row is the coset leader. More specifically, the first row of the standard array is $\mathbf{0} + C$, with $\mathbf{0}$, the coset leader, listed first. We next pick a lowest-weight vector of $\mathbb{F}_q^n$ that is not in C and put it as a coset leader for the second row. The coset in the second row is obtained by adding its leader to each of the words in the first row. This continues until all cosets of C are entered as rows.

The standard array for Example 3.26 is:

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

We decode $\mathbf{y}$ as the codeword at the top of the column containing $\mathbf{y}$. The error vectors that will be corrected are precisely the coset leaders. By choosing a minimum weight vector in each coset as coset leader we ensure that standard array decoding is a nearest neighbor decoding scheme.

Since the code in the above example has minimum distance equal to 2, we can not even correct all single errors. In fact, this code corrects all single errors that occur within the first 3 positions, but it does not correct errors that occur in the fourth position.

Question 3.27. Explain why the above code, with the above decoding scheme, can not correct errors in the fourth position.

Syndrome decoding is a related decoding scheme for linear codes that uses the parity check matrix H of a code. Suppose $\mathbf{x}$ is transmitted and $\mathbf{y}$ is received. Compute $\mathbf{y}H^T$, which we call the *syndrome* of $\mathbf{y}$ and write $S(\mathbf{y})$. We know from the definition of a parity check matrix that $\mathbf{y}H^T = \mathbf{0}$ if and only if $\mathbf{y} \in C$. So, if $S(\mathbf{y}) = \mathbf{y}H^T = \mathbf{0}$, then we conclude that most likely no errors occurred. (It is possible that sufficiently many errors occurred to change the transmitted codeword into a different codeword, but we can not detect or correct this.) If $\mathbf{y}H^T \neq \mathbf{0}$, then we know that at least one error occurred, and so $\mathbf{y} = \mathbf{x} + \mathbf{e}$, where $\mathbf{x}$ is a codeword and $\mathbf{e}$ is an error vector. Since $\mathbf{x}H^T = 0$, we see that $\mathbf{y}H^T = (\mathbf{x} + \mathbf{e})H^T = \mathbf{x}H^T + \mathbf{e}H^T = \mathbf{e}H^T$. So the syndrome of the received

word $\mathbf{y}$ is equal to the syndrome of the error vector $\mathbf{e}$ that occurred during transmission. It is known that $S(\mathbf{u}) = S(\mathbf{v})$ if and only if $\mathbf{u}$ and $\mathbf{v}$ are in the same coset of C. It follows that there is a one-to-one correspondence between cosets and syndromes. This means that every word in a particular coset (*i.e.* in a particular row of the standard array) has the same syndrome. Thus, we can extend the standard array by listing the syndromes of each coset leader (and hence of each element in the coset) in an extra column.

Example 3.28. When the standard array of Example 3.26 is expanded to allow for syndrome decoding, it looks as follows:

0000	1011	0101	1110	00
1000	0011	1101	0110	11
0100	1111	0001	1010	01
0010	1001	0111	1100	10

The syndrome decoding algorithm works as follows. When a vector $\mathbf{y}$ is received, calculate the syndrome $S(\mathbf{y}) = \mathbf{y}H^T$. Locate the syndrome in the table, which requires looking through only one column. Decode $\mathbf{y}$ as $\mathbf{y} - \mathbf{e}$ where $\mathbf{e}$ is the coset leader in the row containing $S(\mathbf{y})$. This is equivalent to decoding $\mathbf{y}$ as the codeword at the top of the column containing $\mathbf{y}$.

Using syndromes saves storage space and time. Instead of storing the entire standard array, we need only store the error vectors and the syndromes of the error vectors for each error vector that we hope to detect. Also, when n is large, locating a received vector within a standard array (as compared to locating a syndrome within the syndrome column) can be difficult and time consuming.

Example 3.29. Suppose that we are decoding using the expanded standard array shown in Example 3.28. Suppose 1111 is received, and compute S(1111) = 01. We find this syndrome in the third row of the syndrome column. The coset leader for this row is 0100, and so we decode 1111 as $1111 - 0100 = 1011$.

Theorem 3.30. For a binary code, the syndrome of a received codeword is equal to the sum of the columns of H that correspond to the positions where errors occurred.

Proof. Let $\mathbf{r}$ be a received codeword with syndrome $S(\mathbf{r}) = \mathbf{r}H^T$. Since $\mathbf{r} = \mathbf{c}+\mathbf{e}$, where $\mathbf{c}$ is a valid codeword and $\mathbf{e}$ is an error vector, we have

$$\begin{aligned} S(\mathbf{r}) &= \mathbf{r}H^T && (9)\\ &= (\mathbf{c}+\mathbf{e})H^T && (10)\\ &= \mathbf{c}H^T + \mathbf{e}H^T && (11)\\ &= \mathbf{0} + \mathbf{e}H^T && (12)\\ &= \mathbf{e}H^T && (13) \end{aligned}$$

Notice that (12) follows from the fact that H is the parity check matrix for C. Since $\mathbf{e}H^T$ is a combination of the columns in H that correspond to the positions where errors occurred, this proves the result. □

3.3 Hamming codes

The fact that the syndrome of a received vector is equal to the sum of the columns of the parity check matrix H where errors occurred gives us some insight about how to construct a parity check matrix for a binary code that will undergo coset or syndrome decoding. First, the columns of H should all be nonzero, since otherwise an error in the corresponding position would not affect the syndrome and would not be detected by the syndrome decoder. Second, the columns of H should be distinct, since if two columns were equal, then errors in those two positions would be indistinguishable.

We use these observations to build a family of codes known as the binary *Hamming codes*, H_r, $r \geq 2$. Any parity check matrix for the Hamming code H_r has r rows, which implies that each column of the matrix has length r. There are precisely $2^r - 1$ nonzero binary vectors of length r, and in order to construct a parity check matrix of H_r, we use all of these $2^r - 1$ column vectors.

Definition 3.31. A binary *Hamming code* H_r of length $n = 2^r - 1$, $r \geq 2$, has parity check matrix H whose columns consist of all nonzero binary vectors of length r, each used once. This gives an $[n = 2^r - 1, k = 2^r - 1 - r, d = 3]$ linear code.

Question 3.32. How many errors can a Hamming code correct?

One parity check matrix for the binary $[7, 4, 3]$ Hamming code H_3, where columns are taken in the natural order of increasing binary numbers is as follows:

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

We now rearrange the columns to get a parity check matrix H in standard form:

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

A generator matrix G in standard form is:

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

Question 3.33. Encode the messages 0000 and 1010 using G. Check that the resulting codewords are valid by using H.

Question 3.34. Write down a parity check matrix for the binary Hamming code with $r = 4$.

It is easy to decode Hamming codes, which are used when we expect to have zero or one error per codeword. If we receive the vector $\mathbf{y}$, compute the syndrome $S(\mathbf{y})$. If $S(\mathbf{y}) = \mathbf{0}$, then assume that $\mathbf{y}$ was the codeword sent. If $S(\mathbf{y}) \neq \mathbf{0}$, then, assuming a single error, $S(\mathbf{y})$ is equal to the column of H that corresponds to the coordinate of $\mathbf{y}$ where the error occurred. Find the column of H that matches $S(\mathbf{y})$, and then correct the corresponding coordinate of $\mathbf{y}$. This decoding scheme is further simplified if the columns of H are arranged in order of increasing binary numbers.

Example 3.35. In Section 2.2, we studied extended codes, which are built from existing codes by adding an overall parity check bit. This example shows a parity check matrix of the extended code of the binary Hamming code H_3:

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

Notice that the last row of the extended parity check matrix gives an overall parity-check equation on the codewords: $x_0 + x_1 + \cdots + x_n = 0$. Compare this parity check matrix with the parity check for the corresponding Hamming code.

3.4 Problems

1. Show that if C is a binary linear code, then the extended code obtained by adding an overall parity check to C is also linear.

2. Prove that either all of the codewords in a binary linear code have even weight or exactly half have even weight.

3. Let C be a length n linear code over $\mathbb{F}_2$. If $\mathbf{v} \in \mathbb{F}_2^n$, but $\mathbf{v}$ is not a codeword in C, show that $C \cup (\mathbf{v}+C)$ is a linear code. State and prove the analogous result for linear codes over the field $\mathbb{F}_q$.

4. Let C be a ternary linear code with generator matrix $\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix}$. List the codewords of C. What is the minimum distance of C? Is C a perfect code?

5. Prove that for any linear code, $C = (C^{\perp})^{\perp}$.

6. Construct a standard array for a binary code having the following generator matrix: $\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$. Decode the received vectors 11111 and 01011. Give examples of (a) two errors occurring in a codeword and being corrected and (b) two errors occurring in a codeword and not being corrected.

7. Construct a syndrome look-up table for the perfect binary $[7, 4, 3]$ code which has generator matrix $\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$. Use the table to decode the following received vectors: 0000011, 1111111, 1100110, and 1010101.

8. Show that a binary code can correct all single errors if and only if any parity check matrix for the code has distinct nonzero columns.

9. Prove that any Hamming code has minimum distance equal to 3.

10. Suppose that C is a binary code with parity check matrix H. Write down the parity check matrix for the extended code of C. (Your new parity check matrix will involve the matrix H.)

11. In this chapter, we defined extended codes of binary codes so that every codeword in the extended code has even Hamming weight. Generalize this idea to define extended codes of q-ary codes. What condition should the weight of each codeword in the extended code satisfy?

12. Let C be an $[n, k]$ binary linear code with generator matrix G. If G does not have a column of zeroes, show that the sum of the weights of all the codewords is $n \cdot 2^{k-1}$. [Hint: How many codewords are there with a 1 in coordinate i for fixed i?] Generalize this to linear codes over the field $\mathbb{F}_q$.

4 Ideals and cyclic codes

This chapter studies more advanced topics from both abstract algebra and coding theory. We begin by introducing *ideals*, which are a commonly studied algebraic construct.

4.1 Ideals

We first studied rings in Chapter 8 of the main text [1]. Recall that we assume rings have identity, 1. We will now study ideals, which are a special type of subset of a ring.

Definition 4.1. A nonempty subset A of a ring R is an *ideal* of R if $a + b \in A$ whenever $a, b \in A$ and $ra, ar \in A$ whenever $a \in A$ and $r \in R$. When R is commutative, $ar = ra$, hence we need only check that $ra \in A$.

We say that an ideal A "absorbs" elements from R.

Example 4.2. For any ring R, $\{0\}$ and R are ideals of R.

Question 4.3. For any positive integer n, prove that the set $n\mathbf{Z} = \{0, \pm n, \pm 2n \ldots\}$ is an ideal of the ring of integers $\mathbf{Z}$.

Question 4.4. Let $\langle 3 \rangle = \{3r | r \in \mathbf{Z}_{36}\}$. Prove that $\langle 3 \rangle$ is an ideal of the ring $\mathbf{Z}_{36}$.

Definition 4.5. Let R be a commutative ring with unity and let $g \in R$. The set $\langle g \rangle = \{rg | r \in R\}$ is an ideal of R called the *principal ideal* generated by g. The element g is called the *generator* of the principal ideal.

So, A is a *principal ideal* if there exists $g \in A$ such that every element $a \in A$ can be written as rg for some $r \in R$.

We studied rings of the form $F[x]/m(x)$, where F is a field, in Chapter 28 of [1]. We continue that study now, and in particular, we focus on ideals of $\mathbb{F}_q[x]/(x^n - 1)$.

Question 4.6. Show that $\langle [x + 1] \rangle$ is an ideal in $\mathbb{F}_2[x]/(x^7 - 1)$.

Theorem 4.7. Let A be an ideal in $\mathbb{F}_q[x]/(x^n - 1)$. The following statements are true:

1. There exists a unique *monic* polynomial $g(x)$ of minimal degree such that $[g(x)] \in A$. (A monic polynomial has a coefficient of 1 on its highest power term.)
2. A is principal with generator $[g(x)]$.
3. $g(x)$ divides $x^n - 1$ in $\mathbb{F}_q[x]$.

Proof. We begin this proof by proving the following claim:

Claim 4.8. Suppose that A is an ideal in $\mathbb{F}_q[x]/(x^n - 1)$, and choose a monic polynomial $g(x)$ of minimal degree such that $[g(x)] \in A$. If $f(x)$ is any polynomial such that $[f(x)] \in A$, then $g(x)$ divides $f(x)$.

Proof of Claim. Write $f(x) = q(x)g(x) + r(x)$, where $\deg r(x) < \deg g(x)$. Then $[f(x)] = [q(x)][g(x)] + [r(x)]$. Since $[f(x)] \in A$ and $[q(x)][g(x)] \in A$, we have $[r(x)] \in A$. So, $[cr(x)] \in A$ for any scalar c. If $r(x) \neq 0$, we can scale it to make it monic, contradicting the minimality of $g(x)$. Hence, $r(x) = 0$ and $g(x)$ divides $f(x)$. □

Items 1, 2, and 3 of Theorem 4.7 all follow from the above claim:

1. Suppose that $h(x)$ is another monic polynomial of minimal degree such that $[h(x)] \in A$. Then $g(x)|h(x)$ by the Claim. Since $g(x)$ and $h(x)$ have the same minimal degree, this implies that $h(x) = cg(x)$. Furthermore, since $h(x)$ and $g(x)$ are monic, this implies that $h(x) = g(x)$. Hence, there is a unique monic polynomial $g(x)$ of minimal degree such that $[g(x)] \in A$.
2. Suppose $[f(x)] \in A$, then $g(x)|f(x)$ by the Claim. Hence $[g(x)]|[f(x)]$, which implies that $[f(x)] = [g(x)][q(x)]$ for some $[q(x)] \in \mathbb{F}_q[x]/(x^n - 1)$. Thus, $A \subseteq \langle [g(x)] \rangle$. Now suppose that $[h(x)] \in \langle [g(x)] \rangle$. Then, $[h(x)] = [g(x)][r(x)]$ for some $[r(x)] \in \mathbb{F}_q[x]/(x^n - 1)$. Since $[g(x)] \in A$, $[r(x)] \in$

$\mathbb{F}_q[x]/(x^n-1)$, the definition of an ideal implies that $[h(x)] \in A$. It follows that $\langle [g(x)] \rangle \subseteq A$. Since we have both inclusions, we conclude that $A = \langle [g(x)] \rangle$, hence A is a principal ideal with generator $[g(x)]$.

3. Note that $[x^n - 1] = 0 \in A$, so by the Claim, $g(x)$ must divide $x^n - 1$.

□

4.2 Cyclic codes

Definition 4.9. An $[n, k, d]$ linear code C is *cyclic* if whenever $(c_0, c_1, \ldots, c_{n-1})$ is a codeword in C, then $(c_{n-1}, c_0, \ldots, c_{n-2})$ is also a codeword in C.

Example 4.10. The binary code $C = \{000, 101, 011, 110\}$ is a cyclic code.

Cyclic codes can be implemented efficiently via simple hardware devices called shift registers. This is of great interest in applications involving fiber optics, where high-speed data rates are possible.

When working with cyclic codes, it is convenient to convert codeword vectors $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ into *code polynomials* $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ of degree less than n. Note that the left-most bit in a codeword is associated with the constant term in the code polynomial.

Question 4.11. What is the corresponding code polynomial for the codeword (1, 0, 1, 1)?

Until now, we have been viewing linear codes as subspaces of $\mathbb{F}_q^n = V(n, q)$. When we use code polynomials to represent codewords, we begin to view cyclic codes as subspaces the ring $\mathbb{F}_q[x]/(x^n - 1)$ of polynomials modulo $x^n - 1$. By working modulo $x^n - 1$, we can achieve a right cyclic shift of a codeword by multiplying the associated code polynomial by x: Consider the code polynomial $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$. Multiplying $c(x)$ by x modulo $x^n - 1$ gives $c'(x) = c_0 x + c_1 x^2 + \cdots + c_{n-1}x^n \equiv c_0 x + c_1 x^2 + \cdots + c_{n-1}$ modulo $x^n - 1$. The codeword associated with $c'(x)$ is $(c_{n-1}, c_0, \ldots, c_{n-2})$, which is clearly the right cyclic shift of the codeword associated with $c(x)$. From now on, we will use the terms "codeword" and "code polynomial" interchangeably. This abuse reflects the fact that you should be thinking about codewords and code polynomials as representing the same thing.

We now use the language of code polynomials to characterize cyclic codes:

Theorem 4.12. A linear code of length n over $\mathbb{F}_q$ is cyclic if and only if C satisfies the following two conditions:

1. $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$
2. $a(x) \in C$ and $r(x) \in \mathbb{F}_q[x]/(x^n - 1) \Rightarrow r(x)a(x) \in C$

Proof. Suppose C is a cyclic code of length n over $\mathbb{F}_q$. Then C is linear, so Condition (1) holds. Now suppose that $a(x) \in C$ and $r(x) = r_0 + r_1 x + \ldots r_{n-1}x^{n-1} \in \mathbb{F}_q[x]/(x^n - 1)$. As discussed above, multiplication of a code polynomial by x corresponds to a cyclic shift of the corresponding codeword. Hence, $xa(x) \in C$. Similarly, $x(xa(x)) = x^2c(x) \in C$, and so on. It follows that $r(x)a(x) = r_0a(x) + r_1xa(x) + \cdots + r_{n-1}x^{n-1}a(x)$ is also in C since each summand is in C. Therefore, Condition (2) also holds.

On the other hand, suppose that Conditions (1) and (2) hold. If we take $r(x)$ to be a scalar, the conditions imply that C is a linear code. Then, if we take $r(x) = x$, Condition (2) implies that C is a cyclic code. □

Theorem 4.12 implies the following:

Corollary 4.13. Cyclic codes of length n over $\mathbb{F}_q$ are precisely the ideals in the ring $\mathbb{F}_q[x]/(x^n - 1)$.

Proof. Suppose C is a cyclic code of length n over $\mathbb{F}_q$. Then, its set of code polynomials is defined in $\mathbb{F}_q[x]/(x^n - 1)$. Since C is a linear code, it is closed under all linear combinations of the code polynomials. Furthermore, since C is a cyclic code, $r(x)a(x) \in C$ for any polynomial $r(x) \in \mathbb{F}_q[x]/(x^n - 1)$. Therefore, C satisfies all conditions of being an ideal in $\mathbb{F}_q[x]/(x^n - 1)$.

On the other hand, suppose that A is an ideal in $\mathbb{F}_q[x]/(x^n - 1)$. Then its elements are polynomials of degree less than or equal to $n - 1$, and the set of polynomials are closed under linear combinations. This shows that the polynomials represent codewords in a linear code. Furthermore, by the definition of an ideal, if $a(x) \in A$, then $r(x)a(x) \in A$ for any polynomial $r(x) \in \mathbb{F}_q[x]/(x^n - 1)$. This implies that A is a cyclic code. □

We can actually say more:

Theorem 4.14. Let C be a q-ary $[n, k]$ cyclic code.

1. In C, there is a unique monic code polynomial $g(x)$ with minimal degree $r < n$ called the *generator polynomial* of C.
2. Every code polynomial $c(x) \in C$ can be expressed uniquely as $c(x) = m(x)g(x)$, where $g(x)$ is the generator polynomial and $m(x)$ is a polynomial of degree less than $(n - r)$ in $\mathbb{F}_q[x]$.
3. The generator polynomial $g(x)$ divides $(x^n - 1)$ in $\mathbb{F}_q[x]$.

Proof. This theorem is a translation of Theorem 4.7 into the language of coding theory. See the proof of Theorem 4.7 above. □

Although each cyclic code contains a unique monic generating polynomial of minimal degree, it may also contain other polynomials that generate the code. These other polynomials are either not monic or not of minimal degree.

The above theorems say a great deal about cyclic codes. We can henceforth think of ideals and cyclic codes as equivalent. More specifically, each cyclic code

C is a principal ideal. Since ideals of $\mathbb{F}_q[x]/(x^n-1)$ are generated by divisors of x^n-1, there are precisely as many cyclic codes of length n as there are divisors of x^n-1 in $\mathbb{F}_q[x]/(x^n-1)$. More formally:

Theorem 4.15. There is a one-to-one correspondence between monic divisors of x^n-1 in $\mathbb{F}_q[x]$ and q-ary cyclic codes of length n.

Proof. By Corollary 4.13 and Theorem 4.14, any q-ary $[n,k]$ cyclic code C is an ideal in $\mathbb{F}_q[x]/(x^n-1)$ with a unique monic generating polynomial $g(x)$ that divides x^n-1. Hence, to any q-ary $[n,k]$ cyclic code, we can associate a unique monic divisor of x^n-1.

One the other hand, suppose $h(x)$ is a monic divisor of x^n-1 in $\mathbb{F}_q[x]$. Consider the code $C=\langle[h(x)]\rangle$. Clearly, the code polynomial $h(x)$ generates C. Suppose however that $h(x)$ is not the generating polynomial. Then C contains some other monic generating polynomial $g(x)$ of minimal degree. Since $[h(x)]\in C$ and $[g(x)]$ is the generating polynomial for C, Claim 4.8 implies that $g(x)$ divides $h(x)$. On the other hand, since $[g(x)]\in C=\langle[h(x)]\rangle$, $[g(x)]$ is of the form $[g(x)]=[h(x)][m(x)]$ for some $[m(x)]\in\mathbb{F}_q[x]/(x^n-1)$. Hence, $h(x)$ divides $g(x)$ modulo (x^n-1). It now follows that $g(x)=h(x)$. Hence, any monic divisor $h(x)$ of x^n-1 is the unique generator polynomial for the cyclic code $C=\langle[h(x)]\rangle$, and there is a one-to-one correspondence between monic divisors of x^n-1 and cyclic codes of length n. □

Theorem 4.15 facilitates listing all of the cyclic codes of a given length.

Example 4.16. In order to find all of the binary cyclic codes of length 3, we must factor x^3-1 into irreducible polynomials $(x+1)(x^2+x+1)$ over $\mathbb{F}_2$. There are four possible generator polynomials: $g_0(x)=1$ generates the code which is equal to all of $\mathbb{F}_2/(x^3-1)$; $g_1(x)=x+1$ generates the code $\{0,1+x,x+x^2,1+x^2\}$; $g_2(x)=x^2+x+1$ generates the code $\{0,1+x+x^2\}$; $g_3(x)=x^3+1$ generates the code $\{0\}$.

Notice that the code $C=\{0,1+x,x+x^2,1+x^2\}$ generated by g_1 in the above example is also generated by x^2+1. However, g_1 is the unique monic generator polynomial of minimal degree.

The code $C=\{0,1+x,x+x^2,1+x^2\}$, can be rewritten as $\{0,\ g_1(x),\ xg_1(x),\ g_1(x)+xg_1(x)\}$. This shows that $g_1(x)=1+x$ and $xg_1(x)=x+x^2$ form a basis for C. This implies that the corresponding vectors (110) and (011) can be used to form a generating matrix for C (recall Section 3.1). In particular, a generating matrix for C has the form:

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

Notice that the first row is the vector corresponding to the generator polynomial $g_1(x)$, and the second row is its cyclic shift. We next show that we can always use the generator polynomial to define the generator matrix of a cyclic code in this way.

Theorem 4.17. Suppose C is a cyclic code with generator polynomial $g(x) = g_0 + g_1x + \cdots + g_rx^r$ of degree r. Then the dimension of C is $n - r$, and a generator matrix for C is the following $(n-r) \times n$ matrix:

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_r & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_r \end{pmatrix}$$

Proof. First, note that g_0 is nonzero: Otherwise, $(0, g_1, \ldots, g_{r-1}) \in C$ which implies that $(g_1, \ldots, g_{r-1}, 0) \in C$ which implies that $g_1 + g_2x + \cdots + g_{r-1}x^{r-1} \in C$, which contradicts the minimality of the degree r of the generating polynomial. Now, we see that the $n-r$ rows of the matrix G are linearly independent because of the echelon of nonzero g_0s with 0s below. These $n - r$ rows represent the code polynomials $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$. In order to show that G is a generator matrix for C we must show that every code polynomial in C can be expressed as a linear combination of $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$. Part 2 of Theorem 4.14 shows that if $c(x)$ is a code polynomial in C, then $c(x) = m(x)g(x)$ for some polynomial $m(x)$ of degree less than $n - r$ in $\mathbb{F}_q[x]$. Hence,

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= (m_0 + m_1x + \cdots + m_{n-r-1}x^{n-r-1})g(x) \\ &= m_0g(x) + m_1xg(x) + \cdots + m_{n-r-1}x^{n-r-1}g(x) \end{aligned}$$

which shows that any code polynomial $c(x)$ in C can be written as a linear combination of the code polynomials represented by the $n-r$ independent rows of G. We conclude that G is a generator matrix for C and the dimension of C is $n - r$. □

Example 4.18. In order to construct a binary cyclic code C of length $n = 7$, we need to find a generator polynomial that is a factor of $x^7 - 1$ in $\mathbb{F}_2[x]$. Choose $g(x) = (1 + x + x^3)(1 + x) = 1 + x^2 + x^3 + x^4$. Since $g(x)$ has degree $r = 4$, it follows that C has dimension $k = n - r = 3$. So our generator matrix G will be a 3×7 matrix:

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

We now have an easy way to write down a generator matrix for a cyclic code. As with linear codes, we can encode a cyclic code by performing a matrix multiplication. However, Part (2) of Theorem 4.14 suggests a different encoding scheme for cyclic codes. We can use the generating polynomial $g(x)$ to encode a message $\mathbf{a} = (a_0, \ldots, a_{k-1}) = a(x)$ by performing the polynomial multiplication

$a(x)g(x)$. This simple polynomial multiplication can be used instead of storing and using an entire generator matrix for the cyclic code.

Question 4.19. Let $g(x) = 1 + x^2 + x^3$ be the generator polynomial for a binary cyclic code of length 7. Encode the message (1, 0, 0, 1) without using a generating matrix.

You may have noticed that the above way of obtaining a generator matrix of a cyclic code gives a matrix that is not in standard form. Therefore, we can not automatically write down the parity check matrix, as we can when we have a generator matrix in standard form. However, we next define *parity check polynomials* that can be used to easily construct parity check matrices.

Definition 4.20. The *parity check polynomial* $h(x)$ for an $[n, k]$ cyclic code C is the polynomial such that $g(x)h(x) = x^n - 1$, where $g(x)$ is the degree r generator polynomial for C. Furthermore, $h(x)$ is monic of degree $k = n - r$.

Since $c(x)$ is a code polynomial if and only if it is a multiple of $g(x)$, it follows that $c(x)$ is a code polynomial if and only if $c(x)h(x) \equiv 0$ modulo $(x^n - 1)$.

Question 4.21. Prove the above statement.

Theorem 4.22. Suppose C is an $[n, k]$ cyclic code with parity check polynomial $h(x) = h_0 + h_1x + \cdots + h_kx^k$. Then, a parity check matrix for C is the following $(n - k) \times n$ matrix:

$$H = \begin{pmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & \cdots & 0 \\ 0 & 0 & h_k & h_{k-1} & \cdots & h_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & h_k & h_{k-1} & \cdots & h_0 \end{pmatrix}$$

Proof. You proved above that if $c(x) \in C$, then $c(x)h(x) \equiv 0$ modulo $(x^n - 1)$. The coefficient of x^j in the product $c(x)h(x)$ is $\sum_{i=0}^{n-1} c_i h_{j-i}$, where the subscripts are taken modulo n. So, if $c(x) \in C$, we have

$$\sum_{i=0}^{n-1} c_i h_{j-i} = 0, \text{ for } 0 \le j \le n-1. \tag{14}$$

The n equations represented in (14) are parity check equations satisfied by the code. The last $n - k$ of these equations imply that if $\mathbf{c} \in C$, then $\mathbf{c}H^T = 0$.

On the other hand, suppose that $\mathbf{c}H^T = 0$. Then the rows of H are vectors in $C^\perp$. Since $h(x)$ is monic, H contains an echelon of 1s with 0s underneath; hence, the $n - k$ rows of H are linearly independent. Since $\dim C = k$, we have $\dim C^\perp = n - k$, so the $n - k$ independent rows of H form a basis for $C^\perp$. It follows that $\mathbf{c} \cdot \mathbf{h} = 0$ for all $\mathbf{h} \in C^\perp$. So $\mathbf{c} \in (C^\perp)^\perp = C$.

Using Definition 3.20 of a parity check matrix, we have shown that H is a parity check matrix for C. □

Example 4.23. The $[7, 3]$ code C constructed in Example 4.18 has parity check polynomial $h(x) = (x^7-1)/g(x) = 1+x^2+x^3$. The following $[(n-k)\times n] = [4\times 7]$ matrix is a parity check matrix for C:

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

Question 4.24. What condition (equation) should the generator matrix from Example 4.18 and parity check matrix from Example 4.23 together satisfy? Check that these matrices do satisfy that equation.

We defined Hamming codes H_r in Section 3.3 by constructing their parity check matrices. We now show that these codes are equivalent to cyclic codes.

Theorem 4.25. The binary Hamming code H_r is equivalent to a cyclic code.

The proof of this theorem assumes knowledge of field theory, as presented in Chapter 28 of the main text [1].

Proof. Let $p(x)$ be an irreducible polynomial of degree r in $\mathbb{F}_2[x]$. Then, the ring $\mathbb{F}_2[x]/p(x)$ of polynomials modulo $p(x)$ is actually a field of order 2^r. Since every finite field has a primitive element (recall the Primitive Root Theorem), there exists an element $\alpha \in \mathbb{F}_2[x]/p(x)$ such that $\mathbb{F}_2[x]/p(x) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^r-2}\}$. Consider the matrix $H = [1, \alpha, \cdots, \alpha^{2^r-2}]$. Since each entry of H is an element of the field $\mathbb{F}_2[x]/p(x)$ of order 2^r, we can express each entry α^i, $0 \le i \le 2^r - 1$ as a binary column vector $(a_{i,0}, a_{i,1}, \ldots, a_{i,r-1})^T$ where $\alpha^i = a_{i,0} + a_{i,1}x + \cdots + a_{i,r-1}x^{r-1}$. This allows us to think of H as an $r \times (2^r - 1)$ binary matrix.

Let C be the linear code having H as its parity check matrix. Since the columns of H are exactly the $2^r - 1$ nonzero binary vectors of length r, C is a length $n = 2^r - 1$ binary Hamming code H_r.

We now show that C is cyclic. Since H is the parity check matrix for C, we have $\mathbf{c} \in C$ if and only if $\mathbf{c}H^T = 0$, *i.e.*,

$$\begin{aligned} C &= \{(c_0, c_1, \ldots, c_{n-1}) \in V(n, 2) \mid c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} = 0\} \\ &= \{c(x) \in \mathbb{F}_2[x]/(x^n - 1) \mid c(\alpha) = 0 \text{ in } \mathbb{F}_2[x]/p(x)\}. \end{aligned}$$

If $c(x) \in C$ and $r(x) \in \mathbb{F}_2[x]/(x^n-1)$, then $r(x)c(x) \in C$ because $r(\alpha)c(\alpha) = r(\alpha) \cdot 0 = 0$. It now follows from Theorem 4.12 that this Hamming code is cyclic. □

4.3 Group of a code

In Chapter 11E of the main text [1], we encountered the group $\mathfrak{S}_3$ of permutations on three symbols. Recall the following:

Definition 4.26. A *permutation* of a set A is a function from A to itself that is both one-to-one and onto.

We will focus on the case where $A = \{0, 1, \ldots, n-1\}$. Here, permutations are rearrangements of the numbers.

Example 4.27. Let $A = \{0, 1, \ldots, 5\}$ and let π be the map from A to A that sends the ordered numbers 0, 1, 2, 3, 4, 5 to the ordered numbers 4, 5, 3, 1, 0, 2. (That is, $\pi(0) = 4$, $\pi(1) = 5$, and so on.) Then π is a permutation of the set A.

We often use *cycle notation* to describe a permutation. For the permutation in Example 4.27, the cycle notation is $\pi = (1\,5\,2\,3)(0\,4)$. The rightmost cycle means that 0 is sent to 4, and 4 is sent to 0. The leftmost cycle means that 1 is sent to 5, 5 is sent to 2, 2 is sent to 3, and 3 is sent to 1. As this example shows, within each cycle, we read left to right. However, since the juxtaposition of cycles denotes composition, we read the rightmost cycle first, and then work leftward. (Some books vary in the convention of which order to read cycles or functions in a composition. We will always read right to left.) For this particular permutation, the order of the cycles does not matter. However, the following example shows that the order of the cycles can make a difference.

Example 4.28. Consider the permutation γ on $\{1, 2, 3\}$ represented by cycle notation $(1\,2\,3)(3\,1)$. To determine $\gamma(2)$, we begin with the rightmost cycle and notice that 2 is not involved in this cycle. Moving leftwards, the next cycle shows that 2 is sent to 3. Since there are no more cycles, we conclude that $\gamma(2) = 3$.

Now consider the permutation γ' represented by cycle notation $(3\,1)(1\,2\,3)$. To determine $\gamma'(2)$, we begin with the rightmost cycle and notice that 2 gets sent to 3. Moving leftwards, we see that 3 gets sent to 1. Since there are no more cycles, we conclude that $\gamma'(2) = 1$. This shows that in general, cycles do not commute.

As you might guess, there are certain conditions under which cycles do commute. This is formalized in the next theorem.

Theorem 4.29. Disjoint cycles commute: If the pair of cycles $a = (a_1\ \ldots\ a_m)$ and $b = (b_1\ \ldots\ b_n)$ have no entries in common, then $ab = ba$.

Proof. See [2]. □

Because disjoint cycles commute, we like to express permutations in terms of disjoint cycles. For example, we can rewrite the permutation $\gamma = (1\,2\,3)(3\,1)$ from Example 4.28 as $(1)(3\,2)$. The cycle (1) means $\gamma(1) = 1$, or in other words, 1 is fixed by the permutation γ. It is customary to omit from the cycle notation the elements that are fixed by the permutation. For the example of γ, we omit the (1) and simply write $\gamma = (3\,2)$. Another example of a permutation that fixes some elements is the permutation ρ on $A = \{0, 1, \ldots, 5\}$ that sends the ordered numbers 0, 1, 2, 3, 4, 5 to the ordered numbers 0, 1, 2, 5, 3, 4. In cycle notation, we write $\rho = (3\,5\,4)$.

Question 4.30. Express the following permutations as products of disjoint cycles:

1. $(1\,2\,3)(3\,4)(5\,0\,2\,1)$
2. $(3\,2)(5\,0\,2)(2\,3\,5)$
3. $(0\,5)(0\,1\,2\,3\,4\,5)$

Definition 4.31. A *permutation group* of a set A is a set of permutations of A that forms a group under function composition.

Above, we use cycle notation and the juxtaposition of cycles to denote the composition of cycles that together describe a particular permutation. When composing cycles from different permutations, we use the exact same procedure. For example, with A, π, and ρ as above, the composition $\phi \circ \rho = \phi\rho$ is written as $(1\,5\,2\,3)(0\,4)(3\,5\,4)$. Since ρ is on the right in the function composition, its cycle is written on the right in the cycle composition. We simplify this product of cycles by writing $\phi\rho$ as a product of disjoint cycles: $(4\,1\,5\,0)(2\,3)$. The function composition $\rho \circ \phi = \rho\phi$ is written as $(3\,5\,4)(1\,5\,2\,3)(0\,4)$, which simplifies into the following disjoint cycles: $(0\,3\,1\,4)(2\,5)$.

In the main text, we worked with the group of permutations on three objects. Above, we gave examples of permutations on six objects. In general, we denote by $\mathfrak{S}_n$ the group of all permutations of n objects.

Question 4.32. Prove that there are $n!$ elements in $\mathfrak{S}_n$.

By now, you should feel comfortable working with cycle notation and cycle composition (function composition). We are ready to show how permutations and permutation groups are connected with coding theory.

We start by showing how permutations of $\mathfrak{S}_n$ can act on vectors (or codewords) of length n. Let $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and let τ be a permutation on $\mathfrak{S}_n$. We can define a new vector $\mathbf{x}\tau$ component-wise by

$$(\mathbf{x}\tau)_i = x_{\tau(i)} \tag{15}$$

In other words, the ith component of $\mathbf{x}\tau$ is equal to the $(\tau(i))$th component of $\mathbf{x}$:

$$\mathbf{x}\tau = (x_0, x_1, \ldots, x_{n-1})\tau = (x_{\tau(0)}, x_{\tau(1)}, \ldots, x_{\tau(n-1)}). \tag{16}$$

For example, for $n = 3$, if $\tau = (0\,2)$ in cycle notation, then

$$\mathbf{x}\tau = (x_0, x_1, x_2)\tau = (x_2, x_1, x_0), \tag{17}$$

because $\tau(0) = 2$, $\tau(1) = 1$, and $\tau(2) = 0$.

Question 4.33. Prove the following "associative law:" For any vector $\mathbf{x}$ and any two permutations $\sigma, \tau \in \mathfrak{S}_n$,

$$(\mathbf{x}\sigma)\tau = \mathbf{x}(\sigma\tau), \tag{18}$$

where $\sigma\tau$ is the product of σ and τ in the group $\mathfrak{S}_n$; recall that this product is the composite $\sigma \circ \tau$, so that $(\sigma\tau)(i) = \sigma(\tau(i))$. [Hint: This problem is easy if you use the definition as given in (15). But you are likely to get confused, and even think that (18) is wrong, if you try to use (16).]

Define σ_n on $A_n = \{0, 1, \ldots, n-1\}$ as the permutation written with cycle notation as $\sigma_n = (0\,1\,\ldots\,n-1)$. Hence $\sigma_n(0) = 1$, $\sigma_n(1) = 2$, ..., $\sigma_n(n-1) = 0$. For convenience, we write $\sigma_n = \sigma$. Consider $\mathbf{x}\sigma$, where $\mathbf{x}$ is any vector of length n:

$$\begin{aligned} \mathbf{x}\sigma &= (x_0, x_1, \ldots, x_{n-1})\sigma && (19)\\ &= (x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(n-1)}) && (20)\\ &= (x_1, x_2, \ldots, x_{n-1}, x_0) && (21) \end{aligned}$$

This shows that $\mathbf{x}\sigma$ is a cyclic shift of $\mathbf{x}$. We can use this notation to restate the definition of a cycle code:

Definition 4.34. A linear code C of length n is cyclic if and only if whenever $\mathbf{c} \in C$, so is $\mathbf{c}\sigma_n$.

Let C be a binary linear $[n, k]$ code. It follows from Definition 3.12 that every permutation of the n coordinates of the codewords of C sends C onto an equivalent $[n, k]$ code or onto itself.

Definition 4.35. The set of all permutations that send a code C onto itself is called the *group of the code* C. It is a group under function composition, and it is denoted $G(C)$.

In other words, $G(C)$ is the group of permutations $\tau \in \mathfrak{S}_n$ such that for any codeword $\mathbf{c} \in C$, the vector $\mathbf{c}\tau$ is also a codeword in C.

Question 4.36. Prove that the group of a code of length n is a subgroup of $\mathfrak{S}_n$.

Example 4.37. If C is the whole space $V(n, q)$, then $G(C) = \mathfrak{S}_n$.

Using the group of a code, we can offer another definition of a cyclic code:

Definition 4.38. A linear code C of length n is cyclic if $G(C)$ contains the cyclic group of order n generated by $\sigma_n = (0\,1\,\ldots\,n-1)$.

For a cyclic code C, $G(C)$ may be, and usually is, larger than the cyclic group of order n generated by $\sigma_n = (0\,1\,\ldots\,n-1)$.

Sometimes we need to find alternate generating matrices for a code, and the group of the code can be used to do this: Any element in $G(C)$ applied to the coordinate positions of any generator matrix of C yields another generator matrix of C. There are several other reasons that coding theorists study groups of codes, for example the group of a code is useful in determining the structure of the code, computing weight distributions, classifying codes, and devising decoding algorithms. We say more about the latter application below.

What follows is a brief discussion on how the group of a code, if it is big enough, can sometimes be used as an aid in decoding. This discussion was written by Professor Ken Brown and is adapted from *The Theory of Error-Correcting Codes* [4, p.513].

Consider an $[n, k, d]$ linear code C with $k \times n$ generator matrix G in standard form, $G = (I \mid A)$. For simplicity, let's assume the code is binary. We then have an $(n-k) \times n$ parity check matrix $H = (A^T \mid I)$. Given a k-bit message $\mathbf{u}$, we encode it as $\mathbf{x}$ by setting

$$\mathbf{x} = \mathbf{u}G = (\mathbf{u} \mid \mathbf{u}A).$$

Since the first k bits of $\mathbf{x}$ contain the message $\mathbf{u}$, we call them the *information bits*; the last $n-k$ bits, given by $\mathbf{u}A$, are called *check bits*.

Assume $d \geq 2t+1$, so that t errors can be corrected. Suppose a codeword $\mathbf{x}$ is sent and an error $\mathbf{e}$ with weight $w(\mathbf{e}) \leq t$ occurs. Then $\mathbf{y} = \mathbf{x} + \mathbf{e}$ is received, and we know in principle that it is possible to decode $\mathbf{y}$ and recover the codeword $\mathbf{x}$. We also know that the syndrome $S(\mathbf{y}) = \mathbf{y}H^T$ can help us do this. The following theorem describes a particularly easy case. Write $\mathbf{e} = (\mathbf{e}' \mid \mathbf{e}'')$, where $\mathbf{e}'$ contains the first k bits of $\mathbf{e}$.

Theorem 4.39. Suppose the syndrome $S(\mathbf{y}) = \mathbf{s}$ has weight $\leq t$. Then $\mathbf{e}' = \mathbf{0}$ (so the k information bits of $\mathbf{y}$ are correct) and $\mathbf{e}'' = \mathbf{s}$. We therefore have

$$\mathbf{x} = \mathbf{y} - (\mathbf{0} \mid \mathbf{s}).$$

Proof. We will prove the equivalent statement that if $\mathbf{e}' \neq \mathbf{0}$ then $w(\mathbf{s}) > t$. Using $H = (A^T \mid I)$ and remembering that $\mathbf{x}H^T = \mathbf{0}$, one computes

$$\mathbf{s} = \mathbf{y}H^T = \mathbf{e}H^T = \mathbf{e}'A + \mathbf{e}''. \tag{22}$$

Consequently,

$$w(\mathbf{s}) \geq w(\mathbf{e}'A) - w(\mathbf{e}''). \tag{23}$$

On the other hand, the vector $\mathbf{e}'G = (\mathbf{e}' \mid \mathbf{e}'A)$ is a nonzero codeword, so it has weight $\geq 2t+1$. Thus $w(\mathbf{e}') + w(\mathbf{e}'A) \geq 2t+1$, or

$$w(\mathbf{e}'A) \geq 2t + 1 - w(\mathbf{e}'). \tag{24}$$

Combining inequalities (23) and (24), we obtain

$$w(\mathbf{s}) \geq 2t + 1 - w(\mathbf{e}') - w(\mathbf{e}'') = 2t + 1 - w(\mathbf{e}) \geq t + 1,$$

as required. □

The converse is also true (and easier): If $\mathbf{e}' = \mathbf{0}$, so that there is no error in the information bits, then the syndrome satisfies $w(\mathbf{s}) \leq t$. To see this, observe that $\mathbf{s} = \mathbf{e}''$ by (22), and $w(\mathbf{e}'') \leq w(\mathbf{e}) \leq t$.

To summarize the discussion so far, decoding is easy if we're lucky enough that the errors don't affect the information bits; moreover, we can tell whether we were lucky by looking at the weight of the syndrome. What if we're unlucky? This is where the group $G(C)$ can help.

Suppose that for every vector $\mathbf{e}$ of weight $\leq t$ there is a permutation $\sigma \in G(C)$ such that the vector $\mathbf{e}\sigma$ obtained by permuting the coordinates according to σ has all 0's in its first k bits. Then we have the following decoding algorithm, always assuming that there are at most t errors: If $\mathbf{y}$ (which equals $\mathbf{x} + \mathbf{e}$) is received, find $\sigma \in G(C)$ such that $\mathbf{y}\sigma$ (which equals $\mathbf{x}\sigma + \mathbf{e}\sigma$) has syndrome of weight $\leq t$. Such a σ exists by our hypothesis and by the discussion above; however, we may have to use trial and error to find it since we don't know $\mathbf{e}$. Then we can decode $\mathbf{y}\sigma$ as explained in the theorem, giving us $\mathbf{x}\sigma$, and then we can recover $\mathbf{x}$ by "unpermuting" the coordinates, *i.e.*, by applying σ^{-1}.

In practice, one tries to find a small subset $P \subseteq G(C)$ such that the permutation σ above can always be found in P; this cuts down the search time. For example, consider the $[7, 4, 3]$ Hamming code, with $t = 1$. Let's take a version of the Hamming code that is cyclic, as we know is possible. Then $G(C)$ contains at least the cyclic group of order 7 generated by the cyclic shift. In this case one can take $P \subseteq G(C)$ to consist of 3 of the elements of the cyclic group. In other words, one can specify 3 "rotations" that suffice to move the one nonzero bit of any vector $\mathbf{e}$ of weight 1 out of the 4 information positions.

4.4 Minimal polynomials

Before we can define the next class of codes that we wish to present (BCH codes), we need to learn how to find *minimal polynomials* of elements in a field. In Chapter 28E of the main text [1], there is an ad hoc method for finding minimal polynomials. Below we give a systematic way to find them. We begin with several definitions.

Definition 4.40. Let α be an element in $\mathbb{F}_{q^m}$. The *minimal polynomial* of α with respect to the subfield $\mathbb{F}_q$ is the monic polynomial $p(x) \in \mathbb{F}_q[x]$ of minimal degree such that $p(\alpha) = 0$.

The above definition depends on the phrase "with respect to the subfield $\mathbb{F}_q$," and it is meaningless without specifying a subfield.

The monic minimal polynomial of an element with respect to a subfield $\mathbb{F}_q$ is unique: Suppose $f(x)$ and $g(x)$ are distinct monic minimal polynomials of $\alpha \in \mathbb{F}_{q^m}$ with respect to $\mathbb{F}_q$. Since $f(x) \neq g(x)$, there exists a nonzero polynomial $r(x)$ such that $f(x) = g(x)+r(x)$, $r(\alpha) = 0$, and $\deg r(x) < \deg f(x)$. This contradicts the minimality of degree of the minimal polynomial.

Question 4.41. Prove that the minimal polynomial of an element is always irreducible.

Example 4.42. If $\beta \in \mathbb{F}_q$, then its minimal polynomial with respect to $\mathbb{F}_q$ is $(x - \beta)$.

Minimal polynomials for elements of infinite fields are defined in an analogous manner. In order to motivate our systematic way of determining minimal polynomials of elements in finite fields, we give some examples of the analogous concepts using the familiar fields of complex and real numbers.

Example 4.43. The minimal polynomial of $i \in \mathbb{C}$ with respect to the subfield $\mathbb{R}$ is $x^2 + 1$.

Example 4.44. Let $\beta = a + bi \in \mathbb{C}$ where $b \neq 0$. The minimal polynomial of β with respect to $\mathbb{R}$ is

$$\begin{aligned}(x-\beta)(x-\overline{\beta}) &= x^2 - (\beta+\overline{\beta})x + \beta\overline{\beta} \\ &= x^2 - 2ax + (a^2+b^2)\end{aligned}$$

Of course, all coefficients of the minimal polynomial are in $\mathbb{R}$. A non-computational reason for why $(x-\beta)(x-\overline{\beta})$ has coefficients in $\mathbb{R}$ is as follows: Complex conjugation is a ring homomorphism $\varphi : \mathbb{C} \to \mathbb{C}$ defined by $\varphi : \beta \mapsto \overline{\beta}$. This induces a ring homomorphism between $\mathbb{C}[x]$ and itself. Since $(x-\beta)(x-\overline{\beta})$ is fixed by the ring homomorphism, it must have coefficients in $\mathbb{R}$. We can rewrite the minimal polynomial of β as $(x-\beta)(x-\varphi(\beta))$.

When working over finite fields, we would like to use some homomorphism in a way that is analogous to our use of conjugation in the case involving $\mathbb{C}$ and its subfield $\mathbb{R}$. Suppose that we are dealing with $\alpha \in \mathbb{F}_{q^m}$ and the subfield $\mathbb{F}_q$. The analog of complex conjugation is the homomorphism $\phi : \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$ defined by $\phi : \alpha \mapsto \alpha^q$. This map fixes precisely the subfield $\mathbb{F}_q$. We might guess that the minimal polynomial of an element $\alpha \in \mathbb{F}_{q^m}$ with respect to $\mathbb{F}_q$ takes the form $(x-\alpha)(x-\phi(\alpha))$. However, this needs a slight modification. For complex conjugation, notice that $\varphi(\varphi(\beta)) = \beta$, so it would be redundant to include the factor $(x-\varphi^2(\beta))$ in the minimal polynomial for $\beta \in \mathbb{C}$. However, in the finite field case, $\phi^2(\alpha) = \alpha^{q^2}$, which is generally not equal to α. In general, the minimal polynomial of $\alpha \in \mathbb{F}_{q^m}$ with respect to $\mathbb{F}_q$ has the form

$$(x-\alpha)(x-\phi(\alpha))(x-\phi^2(\alpha))\cdots = (x-\alpha)(x-\alpha^q)(x-\alpha^{q^2})\cdots$$

The polynomials terminate after the factor $(x-\alpha^{q^{v-1}})$ where v is the smallest integer such that $\phi^v(\alpha) = \alpha$. We make all of these ideas more formal below.

Definition 4.45. Let α be an element in $\mathbb{F}_{q^m}$. The *conjugates* of α with respect to the subfield $\mathbb{F}_q$ are the elements α, α^q, $\alpha^{q^2}, \ldots$.

Again, the above definition depends on the phrase "with respect to the subfield $\mathbb{F}_q$," and it is meaningless without specifying a subfield.

Example 4.46. If $\alpha \in \mathbb{F}_q$, then its only conjugate with respect to $\mathbb{F}_q$ is itself.

Definition 4.47. The *conjugacy class* of $\alpha \in \mathbb{F}_{q^m}$ with respect to the subfield $\mathbb{F}_q$ is the set consisting of the distinct conjugates of α with respect to $\mathbb{F}_q$. Note that α is always contained in its conjugacy class.

Theorem 4.48. The conjugacy class of $\alpha \in \mathbb{F}_{q^m}$ with respect to the subfield $\mathbb{F}_q$ contains v elements, where v divides m and v is the smallest integer such that $\alpha^{q^v} = \alpha$.

Proof. See [8]. □

Example 4.49. Let α be an element of order 3 in $\mathbb{F}_{16}$. The conjugates of α with respect to $\mathbb{F}_2$ are α, α^2, $\alpha^{2^2} = \alpha^{3+1} = \alpha$, etc. Hence, the conjugacy class with respect to $\mathbb{F}_2$ of α is $\{\alpha, \alpha^2\}$.

Question 4.50. Convince yourself that the conjugacy class with respect to $\mathbb{F}_4$ of α, an element of order 63 in $\mathbb{F}_{64}$, is $\{\alpha, \alpha^4, \alpha^{16}\}$.

Theorem 4.51. Let α be an element in $\mathbb{F}_{q^m}$. Let $p(x)$ be the minimal polynomial of α with respect to $\mathbb{F}_q$. The roots of $p(x)$ are exactly the conjugates of α with respect to $\mathbb{F}_q$.

Proof. See [8]. □

Example 4.52. In Example 2 on page 419 of the main text [1], we constructed $\mathbb{F}_8$ by identifying $\mathbb{F}_2[x]/(x^3 + x + 1)$ with $\mathbb{F}_2[\alpha]$, where $\alpha^3 + \alpha + 1 = 0$. Below, we arrange the eight elements of this field into conjugacy classes, using their exponential representations, and list their associated minimal polynomials. The minimum polynomial for the element α^i is denoted $p_i(x)$, and the minimum polynomial for the element 0 is denoted $p_*(x)$.

Conjugacy Class	:	**Associated Minimal Polynomial**
$\{0\}$	:	$p_*(x) = (x - 0) = x$
$\{\alpha^0 = 1\}$	:	$p_0(x) = (x - 1) = x + 1$
$\{\alpha, \alpha^2, \alpha^4\}$	:	$p_1(x) = p_2(x) = p_4(x)$
	=	$(x - \alpha)(x - \alpha^2)(x - \alpha^4) = x^3 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^5\}$	:	$p_3(x) = p_6(x) = p_5(x)$
	=	$(x - \alpha^3)(x - \alpha^6)(x - \alpha^5) = x^3 + x^2 + 1$

Note that simplification above is done using the relation $\alpha^3 = \alpha + 1$ and the fact that addition and subtraction are equivalent in fields of characteristic 2.

Following the method of the above example, it is now easy to find the minimal polynomial of any field element α with respect to a certain subfield. We simply find the appropriate conjugates of α and then immediately form the minimal polynomial by multiplying together factors $(x - \beta)$, where β runs through all conjugates of α (including α itself).

4.5 BCH and Reed-Solomon codes

BCH and Reed-Solomon codes are among the most powerful and most often used algebraic error-control codes. Binary BCH codes were introduced by R. C. Bose and D. K. Ray-Chaudhuri (1960) and independently by A. Hocquenghem (1959). Gorenstein and Zierler later extended the concepts of BCH codes to arbitrary finite fields (1961). Reed-Solomon (RS) codes were first introduced

by Irving Reed and Gustave Solomon in a five-page paper, "Polynomial Codes over Certain Finite Fields," in 1960 [6], and they were given their present name by W. Wesley Peterson in 1961 [5].

BCH codes are cyclic codes whose generator polynomial satisfies a certain condition that guarantees a certain minimum distance:

Definition 4.53. Fix an integer n. Let $\mathbb{F}_{q^m}$ be the smallest extension field of $\mathbb{F}_q$ that contains an element of order n. Let β be an element of $\mathbb{F}_{q^m}$ of order n. A cyclic code of length n over $\mathbb{F}_q$ is called a *BCH code of designed distance* δ if its generator polynomial $g(x) \in \mathbb{F}_q[x]$ is the least common multiple of the minimal polynomials of $\beta^l, \beta^{l+1}, \ldots, \beta^{l+\delta-2}$, where $l \geq 0$ and $\delta \geq 1$.

In other words, the generator polynomial of a BCH code with designed distance δ has as roots $\delta - 1$ consecutive powers of β. When forming a BCH code C, we usually take $l = 1$ in the above definition. In this case, we say that C is a *narrow-sense* BCH code. If $n = q^m - 1$ for some positive integer m, then β is a primitive element of $\mathbb{F}_{q^m}$, and we say that C is a *primitive* BCH code.

It is natural to wonder about the minimum distance of a BCH code. Our next theorem will bound the minimum distance in terms of the designed distance, but first, we need some background on *Vandermonde* matrices.

Definition 4.54. An $n \times n$ *Vandermonde* matrix V is defined as follows:

$$V = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \alpha_1^3 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^3 & \cdots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \alpha_n^3 & \cdots & \alpha_n^{n-1} \end{pmatrix},$$

where the α_i are elements from any finite or infinite field. The transpose of this matrix is also called Vandermonde.

If the α_i are distinct, then the determinant of V is nonzero. In particular, the determinant is equal to $\prod_{j=1}^{n-1} \prod_{i=j+1}^{n} (\alpha_i - \alpha_j)$.

To prove that the determinant of V is nonzero, consider all polynomials $p(t)$ of degree $n-1$. Given any n values $b_1, \ldots, b_n$, there exists a polynomial of degree $n-1$ interpolating these values: $p(t_i) = b_i$ for $i \leq i \leq n$. We can express this as the following matrix equation:

$$\begin{pmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^{n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The coefficient matrix A is clearly Vandermonde. Since the only polynomial of degree $n-1$ with n roots is the zero polynomial, the equation $A\mathbf{x} = \mathbf{0}$ has only the solution $\mathbf{x} = \mathbf{0}$. This implies that A is nonsingular. It follows that Vandermonde matrices have nonzero determinant.

Alternately, one can derive the expression for the determinant by showing that the determinant of V is equal to the determinant of

$$\begin{pmatrix} (\alpha_2-\alpha_1) & (\alpha_2-\alpha_1)\alpha_2 & (\alpha_2-\alpha_1)\alpha_2^2 & \cdots & (\alpha_2-\alpha_1)\alpha_2^{n-2} \\ (\alpha_3-\alpha_1) & (\alpha_3-\alpha_1)\alpha_3 & (\alpha_3-\alpha_1)\alpha_3^2 & \cdots & (\alpha_3-\alpha_1)\alpha_3^{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ (\alpha_n-\alpha_1) & (\alpha_n-\alpha_1)\alpha_n & (\alpha_n-\alpha_1)\alpha_n^2 & \cdots & (\alpha_n-\alpha_1)\alpha_n^{n-2} \end{pmatrix}.$$

The result follows by induction. (Finish it off!)

We will use the fact that Vandermonde matrices are nonsingular in the proof of the following theorem.

Theorem 4.55. *(BCH bound)* Let C be a BCH code of designed distance δ, with all notation as defined in Definition 4.53. Then the minimum distance of C is at least δ.

Proof. For any $c(x) \in C$, we have

$$c(\beta^l) = c(\beta^{l+1}) = \cdots = c(\beta^{l+\delta-2}) = 0,$$

since $c(x)$ is a multiple of $g(x)$. In matrix notation, this says that

$$Hc^T = \begin{pmatrix} 1 & \beta^l & \beta^{2l} & \cdots & \beta^{(n-1)l} \\ 1 & \beta^{l+1} & \beta^{2(l+1)} & \cdots & \beta^{(n-1)(l+1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \beta^{l+\delta-2} & \beta^{2(l+\delta-2)} & \cdots & \beta^{(n-1)(l+\delta-2)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \mathbf{0}$$

Suppose that some $\mathbf{c} \in C$ has weight $w \leq d-1$. Then $c_i \neq 0$ if and only if $i \in \{a_1, a_2, \ldots, a_w\}$ for some indices $\{a_1, \ldots, a_w\} \subseteq \{0, 1, \ldots n-1\}$. Then, $Hc^T = \mathbf{0}$ implies that in particular,

$$\begin{pmatrix} \beta^{a_1 l} & \cdots & \beta^{a_w l} \\ \beta^{a_1(l+1)} & \cdots & \beta^{a_w(l+1)} \\ \vdots & \vdots & \vdots \\ \beta^{a_1(l+\delta-2)} & \cdots & \beta^{a_w(l+\delta-2)} \end{pmatrix} \begin{pmatrix} c_{a_1} \\ c_{a_2} \\ \vdots \\ c_{a_w} \end{pmatrix} = \mathbf{0}$$

Since $w \leq d-1$, we can extract a $w \times w$ matrix from above and get the following equation:

$$\begin{pmatrix} \beta^{a_1 l} & \cdots & \beta^{a_w l} \\ \beta^{a_1(l+1)} & \cdots & \beta^{a_w(l+1)} \\ \vdots & \vdots & \vdots \\ \beta^{a_1(l+w-1)} & \cdots & \beta^{a_w(l+w-1)} \end{pmatrix} \begin{pmatrix} c_{a_1} \\ c_{a_2} \\ \vdots \\ c_{a_w} \end{pmatrix} = \mathbf{0}$$

Since $c_{a_1}, \ldots, c_{a_w}$ are nonzero, the determinant of the matrix on the left is zero. This is $\beta^{(a_1+\cdots+a_w)b}$ times the determinant of the following matrix:

$$H' = \begin{pmatrix} 1 & \cdots & 1 \\ \beta^{a_1} & \cdots & \beta^{a_w} \\ \vdots & \vdots & \vdots \\ \beta^{a_1(w-1)} & \cdots & \beta^{a_w(w-1)} \end{pmatrix}$$

Since H' is a Vandermonde matrix, it has a nonzero determinant, and we have a contradiction. We conclude that the weight of each nonzero codeword $\mathbf{c} \in C$ is at least δ. Hence, by the linearity of C and Theorem 3.5, it follows that the minimum distance of C is at least δ. □

In order to construct a t-error correcting BCH code of length n over $\mathbb{F}_q$, follow the procedure described below.

1. Find an element β of order n in a field $\mathbb{F}_{q^m}$, where m is minimal.
2. Select $(\delta - 1) = 2t$ consecutive powers of β, starting with β^l for some non-negative integer l.
3. Let $g(x)$ be the least common multiple of the minimal polynomials for the selected powers of β with respect to $\mathbb{F}_q$. (Each of the minimal polynomials should appear only once in the product).

Example 4.56. In this example, we build two binary BCH codes of length 31. Since $31 = 2^5 - 1$, we can find a primitive element of order 31 in $\mathbb{F}_{32}$, and our BCH codes will be primitive. In particular, take β to be a root of the irreducible polynomial $x^5 + x^2 + 1$.

One can generate the following chart of conjugacy classes of elements in $\mathbb{F}_{32}$ and their associated minimal polynomials.

Conjugacy Class	:	**Associated Minimal Polynomial**
$\{0\}$	:	$p_*(x) = (x-0) = x$
$\{\alpha^0 = 1\}$	:	$p_0(x) = (x-1) = x+1$
$\{\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}\}$	:	$p_1(x)$
	=	$(x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)(x-\alpha^{16})$
	=	$x^5 + x^2 + 1$
$\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{17}\}$	:	$p_3(x)$
	=	$(x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^{24})(x-\alpha^{17})$
	=	$x^5 + x^4 + x^3 + x^2 + 1$
$\{\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^9, \alpha^{18}\}$	:	$p_5(x)$
	=	$(x-\alpha^5)(x-\alpha^{10})(x-\alpha^{20})(x-\alpha^9)(x-\alpha^{18})$
	=	$x^5 + x^4 + x^2 + x + 1$
$\{\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{25}, \alpha^{19}\}$	:	$p_7(x)$
	=	$(x-\alpha^7)(x-\alpha^{14})(x-\alpha^{28})(x-\alpha^{25})(x-\alpha^{19})$
	=	$x^5 + x^4 + x^2 + x + 1$
$\{\alpha^{11}, \alpha^{22}, \alpha^{13}, \alpha^{26}, \alpha^{21}\}$	:	$p_{11}(x)$
	=	$(x-\alpha^{11})(x-\alpha^{22})(x-\alpha^{13})(x-\alpha^{26})(x-\alpha^{21})$
	=	$x^5 + x^4 + x^3 + x + 1$
$\{\alpha^{15}, \alpha^{30}, \alpha^{29}, \alpha^{27}, \alpha^{23}\}$	:	$p_{15}(x)$
	=	$(x-\alpha^{15})(x-\alpha^{30})(x-\alpha^{29})(x-\alpha^{27})(x-\alpha^{23})$
	=	$x^5 + x^3 + 1$

To construct a binary cyclic code, its generator matrix $g(x)$ is formed by one or more of the above minimal polynomials. If we want to form a t-error correcting BCH code, then $g(x)$ must have as zeros $2t$ consecutive powers of β.

Let's first construct a 1-error-correcting narrow-sense primitive BCH code. This implies that $l = 1$, $\delta = 3$, and $g(x)$ must have β and β^2 as zeros. Since $p_1(x)$ is the minimal polynomial for both β and β^2, we have simply that $g(x) = p_1(x) = x^5 + x^2 + 1$. Since $\deg g(x) = 5$, the dimension of our BCH code is $31 - 5 = 26$. Therefore, we have defined a $[31, 26]$ binary 1-error-correcting narrow-sense primitive BCH code.

Now let's construct a 2-error-correcting narrow-sense primitive BCH code. This implies that $l = 1$, $\delta = 5$, and $g(x)$ must have β, β^2, β^3, and β^4 as zeros. Since $p_1(x) = p_2(x) = p_4(x)$, the least common multiple of $p_1(x)$, $p_2(x)$, $p_3(x)$, and $p_4(x)$ is equal to the least common multiple of $p_1(x)$ and $p_3(x)$. One can show that the least common multiple of $p_1(x)$ and $p_3(x)$ is equal to their product, $x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$. Since $\deg g(x) = 10$, we have constructed a $[31, 21]$ 2-error-correcting narrow-sense primitive BCH code.

For a given level of error-correcting capability, we try to minimize the re-

dundancy of a code in order to maximize the rate of the code. In constructing BCH codes, this translates to trying to minimize the number of extraneous roots in the minimal polynomial. We know from our study of minimal polynomials and conjugacy classes that the extraneous zeros are the conjugates of the desired zeros. We can often choose l carefully so as to avoid including too many extraneous roots.

Reed-Solomon codes are the most used codes today, due to their use in compact disc digital audio systems. They are also well known for their role in providing pictures of Saturn and Neptune during space missions such as the *Voyager*. These and several other applications of RS codes are described in [9]. One way to define RS codes is as a special case of nonbinary BCH codes:

Definition 4.57. A Reed-Solomon code is a q-ary BCH code of length $q-1$ ($q \neq 2$).

Consider the construction of a t-error-correcting Reed-Solomon code of length $q-1$. The first step is to note that the required element β of order $q-1$ can be found in $\mathbb{F}_q$. Hence, the conjugacy class of β with respect to $\mathbb{F}_q$ consists of the unique element β. It follows that all $2t$ consecutive powers of β are also in $\mathbb{F}_q$, and their conjugacy classes with respect to $\mathbb{F}_q$ are also singletons. It follows from Theorem 4.51 that the minimal polynomial for any power β^s is of the form $(x-\beta^s)$. Hence, a t-error-correcting RS code has a generator polynomial of degree $2t$ with no extraneous roots. In particular, a RS code of length $q-1$ and designed distance δ has generator polynomial $g(x) = (x-\beta^l)(x-\beta^{l+1})\cdots(x-\beta^{l+\delta-2})$ for some $l \geq 0$. If we need to construct a t-error-correcting RS code of length $n = q-1$ over $\mathbb{F}_q$, we define its generator polynomial as

$$g(x) = \prod_{j=0}^{2t-1} (x - \beta^{l+j}).$$

We often take $l = 1$ and form a narrow-sense RS code.

Example 4.58. Let's construct a 2-error-correcting RS code over $\mathbb{F}_8$. This implies that the length is 7. Let β be a root of the irreducible polynomial x^3+x+1, so that β has order 7. The generator polynomial $g(x)$ will have as zeros four consecutive powers of β. No matter how we choose the four consecutive powers of β, there will be no extraneous roots in the generator polynomial. We may as well construct a narrow-sense RS code: $g(x) = (x-\beta)(x-\beta^2)(x-\beta^3)(x-\beta^4) = x^4+\beta^3x^3+x^2+\beta x+\beta^3$. Since $\deg g(x) = 4$, we have constructed a $[7,3]$ 2-error-correcting narrow-sense RS code over $\mathbb{F}_8$.

Theorem 4.59. ([8]) An $[n,k]$ Reed-Solomon code C has minimum distance $d = n-k+1$.

Proof. By the Singleton bound given in Theorem 3.23, we know that $d \leq (n-k+1)$. Since C is an $[n,k]$ code, its generator polynomial has $(n-k)$ roots. Since RS codes of designed distance δ have generator polynomials with $(\delta-1)$ roots, we conclude that $(n-k) = (\delta-1)$. So, $\delta = n-k+1$, and by the BCH

bound, we have $d \geq \delta = n - k + 1$. Now, we can combine both inequalities to see that $d = n - k + 1$. □

Theorem 4.59 shows that RS codes satisfy the Singleton bound of Theorem 3.23 with equality. This means that given their length and dimension, RS codes are optimal in their distance properties. Codes that satisfy the Singleton bound with equality are called *maximum distance separable (MDS)*. One notable property of MDS codes is that if a code is MDS, then so is its dual. (The proof is left as a homework problem.)

4.6 Problems

1. Show that the subring of rational numbers is not an ideal in the Reals.
2. Let R be a commutative ring with unity and let $a \in R$. Prove that the set $\langle a \rangle = \{ra | r \in R\}$ is an ideal of R.
3. Suppose that I is an ideal of R and that $1 \in I$. Prove that $I = R$
4. Show that every ideal in $\mathbf{Z}$ is a principal ideal.
5. List all of the distinct ideals in $\mathbb{F}_2[x]/\langle (x^7 - 1) \rangle$ by listing their generators.
6. List by dimension all of the binary cyclic codes of length 31. Do the same for length 63 and 19.
7. List the cosets of the linear code having the following generator matrix:

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

8. Let C be a q-ary linear code with parity check matrix H. Let $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$. Prove directly that $\mathbf{u}H = \mathbf{v}H$ if and only if $\mathbf{u}$ and $\mathbf{v}$ lie in the same coset of C.
9. Find a basis for the smallest binary linear cyclic code of length 7 containing the codeword 1101000.
10. Let $g(x)$ be the generator polynomial of a binary cyclic code which contains some codewords of odd weight. Is the subset of even-weight codewords in $\langle g(x) \rangle$ a cyclic code? If so, what is the generator polynomial of this subcode?
11. Suppose that a generator matrix G of a linear code C has the property that a cyclic shift of any row of G is also a codeword. Show that C is a cyclic code.

12. If C is an $[n, k]$-cyclic code, show that every k successive positions are information positions, if the first k are.

13. Fix b and prove the following: A BCH code of length n and design distance δ_1 contains as linear subspaces all length n BCH codes with design distance $\delta_2 \geq \delta_1$.

14. Write down a generator polynomial $g(x)$ for a 16-ary $[15, 11]$ Reed–Solomon code.

15. Prove that if C is MDS, then so is its dual code $C^{\perp}$.

16. Prove the following: The minimum distance of the extended code of a RS code, formed by adding a parity check digit to the end of each codeword in the RS code, is increased by 1.

17. We studied the *order* of a group element in Chapter 11B of [1]. If a permutation is written in cycle notation (using disjoint cycles), how can you compute its order? Find the order of the following permutations: $(1\,4)$, $(1\,4\,7\,6\,2)$, and $(1\,2\,3)(4\,5\,7)$.

18. Prove that $\mathfrak{S}_n$ is non-abelian for all $n \geq 3$.

19. Prove that the groups of dual codes satisfy the following equation: $G(C) = G(C^{\perp})$.

5 Acknowledgements

The books in the following bibliography were of great help in writing this packet. We borrowed some examples, problems, and proofs from these books, most often [3] and [8].

The first version of this packet was written in the fall of 2001 at Cornell University, while the author was being supported by an NSF VIGRE grant held by the Mathematics Department. The current version was written during June 2002, while the author was being supported by the Cornell Mathematics Department. This version contains some additional theorems, proofs, examples, and content, and it has been revised to accommodate several suggestions compiled by Ken Brown and Steph van Willigenburg while teaching Math 336 during the Spring 2002 semester. This version also incorporates some additional problems and a handout on the group of a code developed by Ken Brown during the Spring 2002 semester. The author thanks Ken and Steph for their helpful comments and contributions.

References

[1] L. N. Childs, *A concrete introduction to higher algebra*, Springer-Verlag, New York, second ed., 1995.

[2] J. Gallian, *Contemporary Abstract Algebra*, D.C. Heath and Company, Lexington, MA, third ed., 1994.

[3] R. Hill, *A first course in coding theory*, Oxford University Press, New York, first ed., 1986.

[4] F. MacWilliams and N. Sloane, *The Theory of Error Correcting Codes*, North-Hollard Publishing Company, Amsterdam, 1977.

[5] W. W. Peterson, *Error-correcting codes*, The M.I.T. Press, Cambridge, Mass., 1961.

[6] I. S. Reed and G. Solomon, *Polynomial codes over certain finite fields*, J. Soc. Indust. Appl. Math., 8 (1960), pp. 300–304.

[7] J. H. van Lint, *Introduction to coding theory*, Springer-Verlag, Berlin, third ed., 1999.

[8] S. B. Wicker, *Error control systems for digital communication and storage*, Prentice-Hall, Upper Saddle River, first ed., 1995.

[9] S. B. Wicker and V. K. e. Bhargava, *Reed-Solomon codes and their applications*, IEEE Press, Piscataway, first ed., 1994.