Introduction to Algebraic Coding Theory

With Gap

Fall 2006

Sarah Spence Adams*

January 11, 2008

*The first versions of this book were written in Fall 2001 and June 2002 at Cornell University, respectively supported by an NSF VIGRE Grant and a Department of Mathematics Grant. The current version was written in 2005 at Franklin W. Olin College of Engineering supported by NSF CCLI DUE-0410517 Grant. Edits were made in 2006 under the same grant. **If you wish to use this book, please simply let me know and keep me informed of any typos or other problems you find with the book.**

Contents

A Note from the Author

Using This Book

This book serves as a fairly terse introduction to the exciting field of coding theory. You can supplement your reading of this book with any of the books in the bibliography. The more you read, the more you will learn.

The author believes that reading actively is infinitely more productive than reading passively. Accordingly, as you read this book, you will notice *reading questions* interspersed throughout the text. These questions are meant to be straight-forward checks of your reading comprehension. You should work out each of these questions as you read. You can check your answers with the brief solutions provided at the end of each chapter. Be sure to try the questions before looking at the answers!

Most sections incorporate examples using Gap to help students see the power of using software in the study of error-control codes. You should work through the Gap examples to get comfortable with this software.

More challenging homework problems are found at the end of each chapter. Brief solutions to certain homework problems are found at the very end of the book.

This book is a living document. It is recommended that you do not print sections far in advance, as the document is subject to continual improvement. In the electronic version, hyperlinks facilitate the navigation through the book. Constructive suggestions for the improvement of this book are always welcome.

Acknowledgements

The first version of this book was written in the fall of 2001 at Cornell University, supported by an NSF VIGRE grant held by the Mathematics Department. The second version was written during June 2002, supported by the Cornell Mathematics Department. The second version was revised to accommodate several suggestions compiled by Ken Brown and Steph van Willigenburg while teaching Math 336, Applicable Algebra, during the Spring 2002 semester. It also incorporated 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.

The third version of this book was written during 2005, supported by an NSF CCLI A&I Grant. This version expands upon the second version and includes several additional sections designed to make the book appropriate for students without abstract algebra backgrounds. Additional problems have been adapted from the materials developed under NSF DUE-9980900. This version also incorporates extra examples, proofs, and other contributions compiled by the Olin students in the author's Spring 2005 MTH3140, Error Control Coding course. The Gap and Guava examples and exercises were designed by Michael Wu, a student assistant also supported by the NSF CCLI A&I grant. The author thanks Michael for his significant contributions to the Gap and other aspects of the book.

The fourth and current version of this book represents improvements such as solutions to all reading questions, solutions to selected homework problems, general clarifications, and new examples. The author thanks Michael Foss for his detailed suggestions for improvement.

The books in the following bibliography were of great help in writing this book. We borrowed some examples, problems, and proofs from these books, most often [2] and [8].

Sarah Spence Adams
Fall 2006

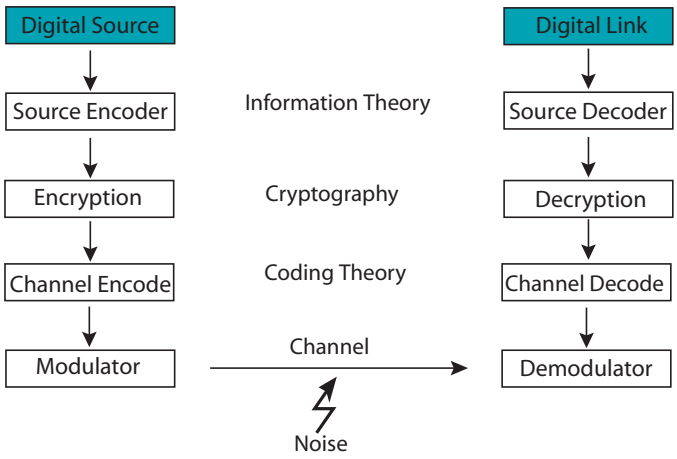

Figure 2: A Flowchart Representing the Various Stages of Data Processing

1 Introduction to Coding Theory

Imagine that you are using an infrared link to beam an mp3 file from your laptop to your PalmPilot. It is possible to model the transmitted data as a string of 0s and 1s. When a 0 is sent, your PalmPilot usually receives a 0. Occasionally, *noise* on the channel, perhaps in the form of atmospheric disturbances or hardware malfunctions, causes the 0 to be received as a 1. We would like to develop ways to combat these errors that can occur during data transmission.

Error-control codes are used to detect and correct errors that occur when data are transmitted across some noisy channel or stored on some medium. 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. 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.

Correcting errors is even more important when transmitting data that have been encrypted for security. In a secure cryptographic system, changing one bit in the ciphertext propagates many changes in the decrypted plaintext. Therefore, it is of utmost importance to detect and correct errors that occur when transmitting enciphered data.

The study of error-control codes is called *coding theory*. This area of discrete applied mathematics includes the study and discovery of various coding schemes that are used to increase the number of errors that can be corrected during data transmission. Coding theory emerged following the publication of Claude Shannon's seminal 1948 paper, "A mathematical theory of communication," [6]. It is one of the few fields that has a defined beginning; you can find a list of historical accomplishments over the last 50 years at

http://www.ittc.ku.edu/ paden/reference/guides/ECC/history.html.

Error control coding is only part of the processing done to messages that are to be transmitted across a channel or stored on some medium. Figure 2 shows a flow chart that illustrates how error control coding fits in with the other stages of data processing.

The process begins with an information source, such as a data terminal or the human voice. The source encoder transforms the source output into a sequence of symbols which we call a message $\mathbf{m}$; if the information source is continuous, the source encoding involves analog-to-digital conversion. Throughout this book, we usually assume that the symbols used are from the binary set $\{0, 1\}$ and call these symbols *bits*. If security is desired, the message would next be encrypted using a

cipher, the subject of the field of cryptography. The next step is the error-control coding, also called channel coding, which involves introducing controlled redundancy into the message $\mathbf{m}$. The output is a string of discrete symbols (usually binary in this book) which we call a codeword $\mathbf{c}$. Next, the modulator transforms each discrete symbol in the codeword into a wavelength to be transmitted across the channel. The transmission is subject to noise of various types, and then the processes are reversed.

Historically, decreasing the error rates in data transmissions was achieved by increasing the power of the transmission. However, this is an inefficient and costly use of power. Moreover, increasing the power only works well on certain channels such as binary symmetric channels; other channels, such as fading channels, present additional challenges. Hence, we will be studying more power- and cost-efficient ways of reducing the error rates. In particular, we will be studying linear, cyclic, BCH, and Reed-Solomon codes.

Through our study of error-control codes, we will model our data as strings of discrete symbols, often binary symbols $\{0, 1\}$. When working with binary symbols, addition is done modulo 2. For example, $1+1 \equiv 0 \pmod 2$. We will study channels that are affected by additive white Gaussian noise, which we can model as a string of discrete symbols that get added symbol-wise to the codeword. For example, if we wish to send the codeword $\mathbf{c} = 11111$, noise may corrupt the codeword so that the $\mathbf{r} = 01101$ is received. In this case, we would say that the error vector is $\mathbf{e} = 10010$, since the codeword was corrupted in the first and fourth positions. Notice that $\mathbf{c} + \mathbf{e} = \mathbf{r}$, where the addition is done component-wise and modulo 2. The steps of encoding and decoding that concern us are as follows:

$$\mathbf{m} \rightarrow \mathit{Encode} \rightarrow \mathbf{c} \rightarrow \mathit{Noise} \rightarrow \mathbf{c} + \mathbf{e} = \mathbf{r} \rightarrow \mathit{Decode} \rightarrow \tilde{\mathbf{m}}$$

where $\mathbf{m}$ is the message, $\mathbf{c}$ is the codeword, $\mathbf{e}$ is the error vector due to noise, $\mathbf{r}$ is the received word or vector, and $\tilde{\mathbf{m}}$ is the decoded word or vector. The hope is that $\tilde{\mathbf{m}} = \mathbf{m}$.

1.1 Introductory Examples

To get the ball rolling, we begin with some examples of error-control codes. Our first example is an error-detecting, as opposed to error-correcting, code. A code that can only *detect* up to t errors within a codeword can be used to determine whether t or fewer errors occurred during the transmission of the codeword, but it cannot tell us exactly what error(s) occurred nor can it fix the error(s). A code that can *correct* up to t errors can be used to actually correct up to t errors that occur during the transmission of a codeword. This means that the code detects that errors occurred, figures out what errors occurred in which positions, and then corrects the errors.

Example 1.1.1. (*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 (bounded between 0 and 9, inclusive) 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 $x_1 x_2 \cdots x_{10}$ satisfies

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

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

The ISBN code can detect any single error and any double-error created by the transposition of two digits. A great course project would be to figure out why this code has this error-detecting capability and compare this with other check-digit schemes used on airplane tickets, in bank numbers on checks, in credit card numbers, in the Universal Product Code (UPC) found on groceries, etc.✓

Question 1. Are either of the following strings valid ISBNs: 0-13165332-6, 0-1392-4101-4?

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. Some of his results and techniques are summarized in

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 1.1.2. (*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. We say that the code's alphabet is the set $\{0, 1\}$, with all arithmetic done modulo 2. Alternatively, we can say that this code alphabet is the *finite field* with two elements, $GF(2)$. Finite fields will be formally defined in Section 2.2. Under certain reasonable assumptions about the channel in use, the receiver decodes a received 5-tuple using a "majority vote": The received 5-tuple is decoded as the bit that occurs most frequently. 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 1.3. ✓

Question 2. How many errors can a binary repetition code of length 10 correct? How many errors can it detect?

Notice that transmitting a message that has been encoded using this repetition code takes five times longer than transmitting the uncoded message. However, we have increased the probability of decoding the received string as the correct message. A complete discussion on this probability is found in Section 1.3.

Guava has a built-in function for creating repetition codes. For example, the command

```
gap> C := RepetitionCode(5, GF(2));;
```

creates a binary repetition code of length 5 and stores it as C. The second argument `GF(2)` indicates that each component of the codewords is in $\{0, 1\}$ and the bitwise operations are performed modulo 2. For now, we will simply look at the generated codewords: The command `AsSortedList` will list all codewords of a code. With `C` defined by our first command, we can now type:

```
gap> cw := AsSortedList(C);
```

which will output

```
[[0000000], [1111111]]
```

Question 3. Use Gap (Guava) to generate a repetition code of length 15 and list its codewords.

Example 1.1.3. (*The 2-D Parity Check Code*) In the ISBN code, we saw that a check digit could be appended to a message in order to *detect* common errors. It is possible to extend this idea to *correct* single errors by first arranging a message into a matrix, or two-dimensional array. For example, suppose we want to encode the 20-bit message 10011011001100101011. First, we arrange this message into a 4×5 matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Then, we look at the *parity* of each row, meaning that we check if there is an even or odd number of 1's in the row. In other words, we check if the sum of the entries in a row is even or odd. Then, we calculate and append a *parity check bit* to the end of each row, where a parity check bit is a bit chosen to ensure that the overall sum of the extended row is even. After appending a parity check bit to each row, we have four rows of length 6. We then repeat the process by calculating and appending parity check bits along along each of the original columns, which gives the following:

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & \spadesuit \end{bmatrix}.$$

The entry in the lower right hand corner marked by ♠ is then calculated as the parity check bit of the new row (in other words, the ♠ entry is calculated as the parity of the parity bits that were calculated along the original columns).

We therefore use the matrix

$$C' = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

to send a stream of 30 bits across our noisy channel.

To see how this extended matrix can correct a single error, suppose that the receiver arranges the received 30 bits into a 5×6 matrix and obtains

$$R = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

The receiver then checks the parity of each row and column: If all parities check to be even, then we assume there are no errors. (Is this a fair assumption?) However, in this example, we see that the parities of the third row and fourth column are odd. Therefore, we conclude that there is an error in the $(3, 4)$ position, correct the error, and proceed to read the message. ✓

Traditionally, the alphabets used in coding theory are finite fields with q elements, $GF(q)$, formally defined in Section 2.2. We say that a code is q-ary if its codewords are defined over the q-ary alphabet

$GF(q)$. The most commonly used alphabets are binary extension fields, $GF(2^m)$. This book focuses on codes with the familiar alphabet $GF(2)$, which are known as *binary codes*. The ISBN code is 11-ary since its symbols come from the set $\{0, 1, \ldots, 10\}$, while our repetition code and 2-D Parity Check Code are both binary.

1.2 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 1.1.2, the code length is 5, since the codewords 00000 and 11111 each contain 5 bits. In this book, we restrict our discussion to *block* codes, which are codes whose codewords are all of the same length. Since error-control codes build redundancy into the messages, the code length, n, is always greater than the original message length, k.

The next parameter that we consider is the *total number of codewords*, M. In Example 1.1.2, 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 a few definitions.

Definition 1.2.1. The *Hamming weight* $w(\mathbf{c})$ of a codeword $\mathbf{c}$ is the number of nonzero components in the codeword.

Example 1.2.2. $w(00000) = 0$, $w(11111) = 5$, $w(1022001) = 4$, $w(1011001) = 4$. ✓

The Guava function `WeightCodeword(C)` can be used to determine the weight of a given codeword:

```
gap> WeightCodeword(Codeword("1011001"));
4
```

Question 4. Use Gap to confirm the weights given in Example 1.2.2.

Definition 1.2.3. The *Hamming distance* between two codewords $d(\mathbf{x}, \mathbf{y})$ is the number of places in which the codewords $\mathbf{x}$ and $\mathbf{y}$ differ. In other words, $d(\mathbf{x}, \mathbf{y})$ is the Hamming weight of the vector $\mathbf{x} - \mathbf{y}$, representing the component-wise difference of the vectors $\mathbf{x}$ and $\mathbf{y}$.

Example 1.2.4. $d(0001, 1110) = 4$, since the two codewords differ in all four positions. $d(1202, 1211) = 2$, since the two codewords differ in the last two positions.

The Guava function `DistanceCodeword` will return the Hamming distance of two codewords. For example:

```
gap> a := Codeword("01111", GF(2));;

gap> b := Codeword("11000",GF(2));;

gap> DistanceCodeword(a,b); 4
```

Question 5. Calculate $d(00111, 11001)$. How about $d(00122, 12001)$? Check your answers using Gap.

Sometimes non-binary codes are evaluated using alternate distance metrics, but in this book, distance will always refer to Hamming distance. To read about other distance functions used to evaluate codes, see [7].

Definition 1.2.5. The *minimum (Hamming) distance* of a code C is the minimum distance between any two codewords in the code: $d(C) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \neq \mathbf{y},\ \mathbf{x}, \mathbf{y} \in C\}$.

Example 1.2.6. The binary repetition code of length 5 has minimum distance 5 since the two codewords differ in all 5 positions.

Question 6. What is the minimum distance between any two ISBN codewords?

Definition 1.2.7. The Hamming distance d is a *metric* on the space of all q-ary n-tuples which means that d satisfies the following properties for any q-ary n-tuples $\mathbf{x}, \mathbf{y}, \mathbf{z}$:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$, with equality if and only if $\mathbf{x} = \mathbf{y}$.
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 property above is called the *triangle inequality*, which should look familiar from Euclidean geometry. The proofs of these properties are left as problems at the end of this chapter.

The notation (n, M, d) is used to represent a code with code length n, a total of M codewords, and minimum distance d. 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).

Example 1.2.8. Let $C = \{0000, 1100, 0011, 1111\}$. Then C is a $(4, 4, 2)$ binary code. ✓

It is possible to construct a code in Gap by listing all of its codewords. To do this, we can use the function `ElementsCode`. For example:

```
gap> C := ElementsCode(["0000", "1100", "0011", "1111"], GF(2));;
```

constructs the code given in Example 1.2.8.

If we need help determining the key parameters for a code, we can use built-in functions such as:

```
gap> n := WordLength(C);
4

gap> M := Size(C);
4

gap> d := MinimumDistance(C);
2
```

The final important code parameter is a measure of efficiency:

Definition 1.2.9. The *rate* of a code is the ratio $\frac{k}{n}$ of message symbols to coded symbols.

Example 1.2.10. The rate of the ISBN code is $\frac{9}{10}$ since each 9-digit message is encoded as a 10-digit message.

Example 1.2.11. The rate of the binary repetition code of length 5 is $\frac{1}{5}$ since each 1-bit message is encoded as a 5-bit message.

Example 1.2.12. The rate of the 2-D Parity Check code $\frac{20}{30}$ or $\frac{2}{3}$ since each 20-bit message is encoded as a 30-bit message.

High rate codes are desirable since a higher rate code implies a more efficient use of redundancy than a lower rate code. In other words, a higher rate code will transmit k message symbols more quickly than a lower rate code can transmit the same k message symbols. However, when choosing a code for a particular application, we must also consider the error-correcting capabilities of the code. A rate 1 code has the optimal rate, but has no redundancy and therefore offers no error control.

1.3 Correcting and Detecting Errors

Talking about the number of errors in a received codeword is equivalent to talking about the distance between the received word and the transmitted word. Suppose a codeword $\mathbf{c} = c_0c_1 \dots c_{n-1}$ is sent through a channel and the received vector is $\mathbf{r} = r_0r_1 \dots r_{n-1}$. The error vector is defined as $\mathbf{e} = \mathbf{r} - \mathbf{c} = e_0e_1 \dots e_{n-1}$. 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. For example, the "majority vote" scheme described for our binary repetition code is an example of a nearest neighbor decoding scheme. 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 (and $1 - p$ of being received correctly)
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 book that the channels involved are symmetric. This implies that error vectors of lower weight will occur with higher *probability* than error vectors of higher weight.

When analyzing codes, we often need to talk about *probabilities* of certain events occurring. For example, we need to compare the probabilities of the occurrences of lower weight errors versus higher weight errors, and we need to ascertain the probability that a codeword will be received in error. Informally, the probability that an event occurs is a measure of the likelihood of the event occurring. Probabilities are numbers between 0 and 1, inclusive, that reflect the chances of an event occurring. A probability near 1 reflects that the event is very likely to occur, while a probability near 0 reflects that the event is very unlikely to occur. When you flip a fair coin, there is a probability of $1/2$ that you will get tails and a probability of $1/2$ that you will get heads.

But what about the probability of getting three heads in three flips of a fair coin? Since each of these events is independent, meaning that each of the three flips does not depend on the other flips, we can use the *multiplicative property* of probabilities: The probability of getting three heads in the three coin tosses is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$.

When dealing with a q-ary codeword of length n, we will often need to ask the following question: What is the probability that e errors occur within a codeword of length n? Let's start with some small examples that work up to this answer.

Example 1.3.1. Given the assumptions of a q-ary symmetric channel with symbol error probability p, consider the probability that no errors will occur when transmitting a q-ary codeword of length n. Since each symbol has probability $(1 - p)$ of being received correctly, the desired probability is $(1 - p)^n$.

Example 1.3.2. Given the assumptions of a q-ary symmetric channel, consider the probability that exactly one of the n symbols in a q-ary codeword of length n is received in error. The probability that the first symbol is received in error and the remaining $n - 1$ symbols are received correctly is $p(1 - p)^{n-1}$. Similarly, the probability that the second is received in error and the remaining $n - 1$ symbols are received correctly is $p(1 - p)^{n-1}$. We can deduce that the overall probability of exactly one of the n symbols in a q-ary codeword of length n being received in error is $np(1 - p)^{n-1}$, since there are n ways that this can happen (namely, one way for each place that the error can occur).

The next logical example should be to ask about the probability that exactly two of the n symbols in a q-ary codeword of length n are received in error. However, we first need a little more experience with counting.

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 $\binom{n}{m} = \frac{n!}{(n-m)!m!}$. 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,

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

Most calculators have a button that will compute binomial coefficients. Let's do some examples to get comfortable with this idea.

Example 1.3.3. How many ways can we choose the chairperson of a committee with four members Alice, Bob, Cathy, and David? The answer should be $\binom{4}{1} = \frac{4!}{3!1!} = 4$. This makes sense since there are clearly 4 ways to choose the chairperson from the group of 4.

Example 1.3.4. How many ways can we choose two places from a codeword of length n? The answer is $\binom{n}{2}$. If $n = 3$, this simplifies to $\binom{3}{2} = 3$. This makes sense since there are 3 ways to *not* choose 1 symbol, or equivalently, 3 ways to choose 2 symbols.

Question 7. By listing all possibilities and by using binomial coefficients, determine how many ways we can choose two places from a codeword of length 4.

We will now use our newfound knowledge on binomial coefficients to resume our earlier discussion.

Example 1.3.5. Given the assumptions of a q-ary symmetric channel, consider the probability that exactly two of the n symbols in a q-ary codeword of length n are received in error. This means that two of the symbols are received in error and $n-2$ of the symbols are received correctly. The probability that the first two symbols are received in error is $p^2(1-p)^{n-2}$. However, there are $\binom{n}{2}$ ways to choose where the 2 errors occur. Therefore, the probability that exactly two of the n symbols in a q-ary codeword of length n are received in error is $\binom{n}{2}p^2(1-p)^{n-2}$.

Question 8. Recall that (assuming a binary symmetric channel), the binary repetition code of length 5 can correct up to two errors. Explain why the probability of decoding a received word correctly is

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

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 1.3.6.
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 cannot 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, $\mathbf{r}$, contains t or fewer errors. Then $d(\mathbf{x}, \mathbf{r}) \leq t$. Let $\mathbf{x}'$ be any codeword other than $\mathbf{x}$. Then $d(\mathbf{x}', \mathbf{r}) \geq t+1$, since otherwise $d(\mathbf{x}', \mathbf{r}) \leq t$ which implies that $d(\mathbf{x}, \mathbf{x}') \leq d(\mathbf{x}, \mathbf{r}) + d(\mathbf{x}', \mathbf{r}) \leq 2t$ (by the triangle inequality), which is impossible since $d(C) \geq 2t+1$. So $\mathbf{x}$ is the nearest codeword to $\mathbf{r}$, and $\mathbf{r}$ is decoded correctly.

□

The remainder of this section will involve using error-correcting capabilities and code rate to compare two different codes proposed for the transmission of photographs from deep-space.

Example 1.3.7. (*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 9. Although the details of the code are not given, use the code's parameters to determine how many errors the (32, 64, 16) Reed-Muller code correct.

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 10. Explain how the expression in (3) was obtained.

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.

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 11. Explain how the expression in (4) was obtained.

Question 12. 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.

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 1.3.7 represents a 6-bit message, the rate of the code is 6/32. Higher rate codes are faster and use less power, however sometimes lower rate codes are preferred because of superior error-correcting capability or other practical considerations.

Question 13. Using code rate and probability of decoding incorrectly to compare the repetition code and the Reed-Muller code, explain which code you think is an overall better code.

1.4 Sphere-Packing Bound

We know from Theorem 1.3.6 that if C has minimum distance $d \geq 2t+1$, then C can correct at least t errors. We also say that C can correct all error vectors of weight t. 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 $\mathbf{c}$ to be surrounded by a sphere of radius t such that the spheres are disjoint (non-overlapping). Any received vector that lies within the sphere centered at $\mathbf{c}$ will be decoded as $\mathbf{c}$. 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 1.4.1. (*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 1.4.1, we need the following lemma.

Lemma 1.4.2. A sphere of radius r, $0 \leq r \leq n$, in the space of all q-ary n-tuples 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 1.4.2. Let $\mathbf{u}$ be a fixed vector in the space of all q-ary n-tuples. 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 $\binom{n}{m}(q-1)^m$. A ball of radius r centered at $\mathbf{u}$ contains vectors whose distance from $\mathbf{u}$ ranges from 0 to r. 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 1.4.1. Suppose C is a t-error-correcting q-ary code of length n. Then, 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 1.4.2. This number of vectors must be less than or equal to the total number of vectors in the space of all q-ary n-tuples, which is q^n. This proves the sphere-packing bound. □

Definition 1.4.3. A code is *perfect* if it satisfies the sphere-packing bound of Theorem 1.4.1 with equality.

Question 14. Show that the binary repetition code of length 5 is perfect.

It is possible to use Gap to test if a code is perfect; the function `IsPerfectCode` return true if a code is perfect and false otherwise. For example, to answer the above question using Gap, we could type

```
gap> C := RepetitionCode(5, GF(2));;

gap> IsPerfectCode(C);
true
```

When decoding using a perfect code, every possible q-ary received word of length n is at distance less than or equal to t from a unique codeword. This implies that the nearest neighbor algorithm yields an answer for every received word $\mathbf{r}$, and it is the correct answer when the number of errors is less than or equal to t. In Chapter 2, we will introduce a family of perfect codes called Hamming codes.

1.5 Answers to Chapter 1 Reading Questions

Answer 1. The first string is valid since the required summation comes out to 198, which is divisible by 11. The second string is not valid since the summation results in 137, which is not divisible by 11.

Answer 2. A binary repetition code of length 10 can correct up to 4 errors since it is a "majority vote"; at 5 errors, there is no way to determine whether 0 or 1 was intended; with 6 or more errors, the received codeword would be "corrected" to the wrong message. However, it can detect 9 errors.

Answer 3. `gap> C := RepetitionCode(15,GF(2));; gap> cw:= AsSortedList(C);`

This will output

```
[[000000000000000],[111111111111111]]
```

Answer 4. `gap> WeightCodeword(Codeword("00000")); 0`

`gap> WeightCodeword(Codeword("11111")); 5`

`gap> WeightCodeword(Codeword("1022001")); 4`

`gap> WeightCodeword(Codeword("1011001")); 4`

Answer 5. $d(00111, 11001)$ is 4. $d(00122, 12001)$ is 5.

Answer 6. For any ISBN codeword, changing a single digit will cause the codeword to no longer be valid since the summation will no longer be divisible by 11. You can, however, change two digits and still follow the ISBN rules, as in **0**-1316-5332-**6** and **1**-1316-5332-**7**. Therefore, the minimum distance between any two ISBN codewords is 2.

Answer 7. By listing all possibilities:

1. first position and second position
2. first position and third position
3. first position and fourth position
4. second position and third position
5. second position and fourth position
6. third position and fourth position

This is a total of 6 ways to choose two place from a codeword of length 4. Similarly, using binomial coefficients, we confirm there are $\binom{4}{2} = \frac{4!}{2!2!} = 6$ ways to pick two places.

Answer 8. This formula gives the probability of getting 0, 1, or 2 errors in a binary repetition code of length 5. $(1-p)^5$ is the probability of getting no errors. $5p(1-p)^4$ is the probability of getting one error. $10p^2(1-p)^3$ is the probability of getting two errors.

Answer 9. The minimum distance of the code is 16, so this code can correct $\lfloor (16-1)/2 \rfloor$, or, 7 errors.

Answer 10. This expression gives the probability of getting 8 or more errors (since the code can correct 7). $\binom{32}{i} p^i (1-p)^{32-i}$ is the probability that there are i errors, where $i \in 8, 9, ..., 32$.

Answer 11. $(1-p)^5 + 5p(1-p)^4 + 10p^2(1-p)^3$ is the probability that a single codeword of length 5 is decoded to the correct message bit. This expression is raised to the 6^{th} power to find the probability that all 6 message bits are decoded correctly.

Answer 12. The probability of decoding incorrectly using the repetition code is 5.9×10^{-5}. The probability of decoding incorrectly using the Reed-Muller code is about 8.5×10^{-10}.

Answer 13. The rate of the Reed-Muller code is 6/32, and the rate of the binary repetition code is 1/5.

Answer 14. The binary repetition code of length 5 satisfies the sphere-packing bound with equality so it is perfect.You can check this using $M = 2$, $n = 5$, and $q = 2$.

1.6 Chapter 1 Problems

Problem 1.1. Complete the ISBN that starts as 0-7803-1025-.

Problem 1.2. Let $C = \{00, 01, 10, 11\}$. Why can't C correct any errors?

Problem 1.3. Let $C = \{(000000), (101110), (001010), (110111), (100100), (011001), (111101), (010011)\}$ be a binary code of length 6.

1. What is the minimum distance of C?
2. How many errors will C detect?
3. How many errors will C correct?

Problem 1.4. Let $A = \mathbb{Z}_5$ and n = 3. Let $C = \{\alpha_1(100) + \alpha_2(011) \mid \alpha_1, \alpha_2 \in \mathbb{Z}_5\}$ where all arithmetic is done componentwise and modulo five. Write down the elements of C and determine the minimum distance of C.

Problem 1.5. Find a way to encode the letters a, b, c, d, e, f, g, h using blocks of zeros and ones of length six so that if two or fewer errors occur in a block, Bob will be able to detect that errors have occurred (but not necessarily be able to correct them).

Problem 1.6. Prove that there is no way to encode the letters a, b, c, d, e, f, g, h using blocks of zeros and ones of length five so that Bob will be able to correct a single error.

Problem 1.7. 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.

Problem 1.8. Prove that the Hamming distance satisfies the following conditions for any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in GF(2)^n$:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$, with equality if and only if $\mathbf{x} = \mathbf{y}$
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
3. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$

Problem 1.9. Is it possible to correct more than one error with the 2-D Parity Check Code shown in Example 1.1.3? How many errors can this code detect?

Problem 1.10. Prove that any ternary $(11, 3^6, 5)$ code is perfect.

Problem 1.11. Show that a q-ary $(q + 1, M, 3)$ code satisfies $M \leq q^{q-1}$.

Problem 1.12. Prove that all binary repetition codes of odd length are perfect codes.

Problem 1.13. Prove that no binary repetition code of even length can be perfect.

Problem 1.14. Prove that for $n = 2^r - 1$, $\binom{n}{0} + \binom{n}{1} = 2^r$.

Problem 1.15. Prove that if C is a binary linear [n, k] code then the sum of the weights of all the elements of C is less than or equal to $n2^{k-1}$.

2 Linear Codes

In this chapter, we study *linear* error-control codes, which are special codes with rich mathematical structure. Linear codes are widely used in practice for a number of reasons. One reason is that they are easy to construct. Another reason is that encoding linear codes is very quick and easy. Decoding is also often facilitated by the linearity of a code. The theory of *general* linear codes requires the use of abstract algebra and linear algebra, but we will begin with some simpler *binary* linear codes

2.1 Binary Linear Codes

Definition 2.1.1. A *binary linear code* C of length n is a set of binary n-tuples such that the componentwise modulo 2 sum of any two codewords is contained in C.

Example 2.1.2. The binary repetition code of length 5 is a binary linear code. You can quickly confirm that it satisfies the requirement that the sum of any two codewords is another codeword: $00000 + 00000 = 00000 \in C$, $00000 + 11111 = 11111 \in C$, and $11111 + 11111 = 00000 \in C$.

Example 2.1.3. The set $\{0000, 0101, 1010, 1111\}$ is a binary linear code, since the sum of any two codewords lies in this set. Notice that this is a (4, 4, 2) code because the codewords have length 4, there are 4 codewords, and the minimum distance between codewords is 2. ✓

Example 2.1.4. The set {0000000, 0001111, 0010110, 0011001, 0100101, 0101010, 0110011, 0111100, 1000011, 1001100, 1010101, 1011010, 1100110, 1101001, 1110000, 1111111} is a binary linear code known as a Hamming code. Notice that this is a (7, 16, 3) code because the codewords have length 7, there are 16 codewords, and the minimum distance is 3. We will study Hamming codes in more detail in Section 2.7. ✓

Question 1. How many errors can the above Hamming code correct?

Question 2. Is $\{000, 100, 001\}$ a binary linear code? How about $\{100, 001, 101\}$?

You can use Gap to check if a given code is linear with the `IsLinearCode` function. For example, to automate the previous question, we can type

```
gap> C := ElementsCode(["000","100", "001"], GF(2));

gap> IsLinearCode(C);
```

and it will output `true` if C is linear and `false` if C is non-linear.

Question 3. Redo Question 2 using Gap.

Question 4. Show that the codeword $\mathbf{0}$ consisting of all zeros is always contained in a binary linear code.

Theorem 2.1.5, below, implies one of the main advantages of linear codes: It is easy to find their minimum distance. This is essential for evaluating the error-correcting capabilities of the code.

Theorem 2.1.5. 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 $d(C) = w(\mathbf{x}-\mathbf{y})$. Note that $\mathbf{x}-\mathbf{y}$ is a codeword in C by the linearity of C. Therefore, since $w^*(C)$ is the weight of the lowest weight codeword, we have $w^*(C) \leq w(\mathbf{x}-\mathbf{y}) = d(C)$.

On the other hand, there exists some codeword $\mathbf{c} \in C$ such that $w^*(C) = w(\mathbf{c})$. By the definition of the weight of a codeword, we can write $w(\mathbf{c}) = d(\mathbf{c}, \mathbf{0})$. Since $\mathbf{c}$ and $\mathbf{0}$ are both codewords, the distance between them must be greater than or equal to the minimum distance of the code: $d(\mathbf{c}, \mathbf{0}) \geq d(C)$. Stringing together these (in)equalities, shows that $w^*(C) \geq d(C)$.

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 2.1.5 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. Notice that the proof does not restrict the linear codes to be binary.

Question 5. 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 linear code and append an *overall parity check digit* to increase the minimum distance of a code. Suppose C is a linear (n, M, d) code. Then we can construct a code C', called the *extended code* of C, by appending a parity check bit 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 2.1.6. 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 2.1.5, $d(C')$ 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 2.1.5, we can conclude that $d(C') = d$ if d is even and $d(C') = d + 1$ if d is odd. □

2.2 Fields, Vector Spaces, and General Linear Codes

In order to develop the theory for general linear codes, we need some abstract and linear algebra. This section may be your first introduction to abstract mathematics. Learning the definitions and getting comfortable with the examples will be very important.

We begin with a discussion of the mathematical constructs known as *fields*. Although the following definition may seem daunting, it will be helpful to keep a couple of simple examples in mind. The binary alphabet, along with modulo 2 addition and multiplication, is the smallest example of a field. A second familiar example is $\mathbb{R}$, the real numbers along with ordinary addition and multiplication.

Definition 2.2.1. Let F be a nonempty set that is *closed* under two binary operations denoted $+$ and $*$. That is, the binary operations input any elements a and b in F and output other elements $c = a + b$ and $d = a * b$ in F. Then $(F, +, *)$ is a *field* if:

1. There exists an *additive identity* labeled 0 in F such that $a + 0 = 0 + a = a$ for any element $a \in F$.
2. The addition is *commutative*: $a + b = b + a$, for all $a, b \in F$.
3. The addition is *associative*: $(a + b) + c = a + (b + c)$, for all $a, b, c \in F$.

4. There exists an *additive inverse* for each element: For each $a \in F$, there exists an element denoted $-a$ in F such that $a + (-a) = 0$.

5. There exists a *multiplicative identity* denoted 1 in F such that $a * 1 = 1 * a = a$ for any element $a \in F$.

6. The multiplication is *commutative*: $a * b = b * a$, for all $a, b \in F$.

7. The multiplication is *associative*: $a * (b * c) = (a * b) * c$, for all $a, b, c \in F$.

8. There exists a *multiplicative inverse* for each nonzero element: For each $a \neq 0$ in F, there exists an element denoted a^{-1} in F such that $a * a^{-1} = a^{-1} * a = 1$.

9. The operations distribute: $a * (b + c) = a * b + a * c$ and $(b + c) * a = b * a + c * a$, for all $a, b, c \in F$.

It is assumed that $0 \neq 1$, so the smallest possible field has two elements (namely, the binary alphabet that we have been using all along). We write this field as $GF(2)$, which means that it is the Galois field (a fancy name for finite fields) with 2 elements. Take a moment to convince yourself that the binary alphabet $GF(2)$ does indeed satisfy all of the field axioms.

Aspiring abstract algebra students will be interested to see that a field is a set whose elements form a commutative *group* under addition (Criteria 1–4), whose nonzero elements form a commutative *group* under multiplication (Criteria 5-8), and whose operations distribute (Criterion 9).

We will write ab as an abbreviation for $a * b$.

In addition to the familiar examples of the real numbers and the binary numbers, it is helpful to have a "non-example" in mind. $(\mathbb{Z}, +, \times)$, the integers under ordinary addition and multiplication, is not a field because most integers do not have multiplicative inverses: For example, there is no integer x such that $2x = 1$. The integers under ordinary addition do form an additive group, but the nonzero integers under ordinary multiplication do not form a multiplicative group.

Example 2.2.2. Let's determine if the set of integers $\mathbb{Z}_6 = \{0, 1, 2, \ldots, 5\}$ is a field under addition and multiplication modulo 6. If the set is a field, there must be a multiplicative inverse for each element of the set. For example, there must exist some k such that $4k \equiv 1 \pmod 6$. However, we know since k is always an integer, $4k$ is always even. Thus, when working modulo 6, $4k$ never be equivalent to 1. Thus, $\mathbb{Z}_6$ is not a field under addition and multiplication modulo 6. Another way to see that this set is not a field is the existence of "zero divisors"; these are non-zero numbers that multiply to zero. For example: $4 \times 3 \equiv 0 \pmod 6$. This would result in a contradiction, since if we multiply each side of this equation on the left by 4^{-1}, we would obtain the nonsensical $3 \equiv 0 \pmod 6$.✓

In contrast to the above example, $\mathbb{Z}_p$ under addition and multiplication modulo p, where p is any prime, is a field. The proof is left as a problem. In fact, it is the field properties of $\mathbb{Z}_{11}$ that give the ISBN code from Example 1.1.1 its error-detecting capabilities.

Question 6. Find additive inverses and multiplicative inverses for the elements in $\mathbb{Z}_7$.

We are now ready to use fields to introduce another algebraic construct:

Definition 2.2.3. Let F be a field. A set V of elements called vectors is a *vector space* if for any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and for any $c, d \in F$:

1. $\mathbf{u} + \mathbf{v} \in V$.

2. $c\mathbf{v} \in V$.

3. $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.
4. $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
5. There exists an element denoted $\mathbf{0} \in V$ such that $\mathbf{u} + \mathbf{0} = \mathbf{u}$.
6. For every element $\mathbf{u} \in V$, there exists an element denoted $-\mathbf{u}$ such that $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$.
7. $c(\mathbf{u} + \mathbf{v}) = c\mathbf{u} = c\mathbf{v}$.
8. $(c + d)\mathbf{u} = c\mathbf{u} + d\mathbf{u}$
9. $c(d\mathbf{u}) = (cd)\mathbf{u}$.
10. $1\mathbf{u} = \mathbf{u}$.

We often say that F is the field of *scalars*. Some familiar vector spaces are $\mathbb{R}^2$ and $\mathbb{R}^3$. In these cases, the vectors are the sets of all real vectors of length 2 or 3, and the fields of scalars are the real numbers.

An important example in coding theory is the vector space of all binary n-tuples. Here, the vectors are the binary n-tuples and the field is $GF(2)$. We say that 0 and 1 are the scalars. We can denote this set as $GF(2)^n$, but when we are thinking of this as a vector space, we normally write $V(n, 2)$, which indicates that our vectors are of length n over the alphabet field of size 2.

We will use the following results on *vector subspaces* more often than the above formal definition of a vector space:

Definition 2.2.4. Let V be a vector space. A subset U of V is called a *subspace* of V if U is itself a vector space under the same operations as V.

Theorem 2.2.5. Let V be a vector space and F be a field. A subset U of V is a subspace if:

1. For any two vectors $\mathbf{u}, \mathbf{v} \in U$, $\mathbf{u} + \mathbf{v}$ is also in U;
2. For any $c \in F$, $\mathbf{u} \in U$, $c\mathbf{u}$ is also in U.

You may recall that $\mathbb{R}^2$ and $\mathbb{R}^3$ are known as 2-dimensional and 3-dimensional spaces, respectively. In order to make this concept more precise, we need to develop some mathematical theory.

Definition 2.2.6. Two vectors are *linearly independent* if they are not scalar multiples of each other. Otherwise, the vectors are called *linearly dependent*.

Example 2.2.7. When working in the familiar vector space of $\mathbb{R}^3$, the vectors [100] and [030] are linearly independent, since they are clearly not scalar multiples of each other. However, [211] and [633] are linearly dependent, since the latter vector is 3 times the former vector. ✓

Definition 2.2.8. Vectors in a set are *linearly independent* if none of the vectors are linear combinations of other vectors in the set. More precisely, the vectors $\{\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_{n-1}}\}$ are linearly independent if the only linear combination $c_0\mathbf{v_0} + c_1\mathbf{v_1} + \cdots + c_{n-1}\mathbf{v_{n-1}}$ that gives 0 is the linear combination where all scalars c_i are equal to 0. Otherwise, the vectors are called *linearly dependent*.

Example 2.2.9. When working in the familiar vector space of $\mathbb{R}^3$, the vectors [100], [010], and [001] are linearly independent, since none of these vectors are linear combinations of the others. More precisely, if $c_0[100] + c_1[010] + c_2[001] = 0$, then the only possible solution to this equation is when $c_0 = c_1 = c_2 = 0$. However, [100], [010], and [550] are linearly dependent since the latter vector is a linear combination of the first two. More precisely, if $c_0[100] + c_1[010] + c_2[550] = 0$, then we can solve this equation with $c_0 = 5$, $c_1 = 5$, and $c_2 = -1$. Since the coefficients of this linear combination are not all zero, the vectors are linearly dependent. (Notice that the abbreviation $[xyz]$ is used for the vector $[x, y, z]$ when no confusion should occur.) ✓

Question 7. Are the following vectors independent in $\mathbb{R}^3$:

1. $\{[1,1,1],[0,3,4],[3,9,11]\}$?
2. $\{[1,0,1],[12,3,0],[13,3,0]\}$?

Definition 2.2.10. A *basis* is a (minimal cardinality) set of linearly independent vectors that span the vector space.

It follows from this definition that any vector in the vector space is a linear combination of the basis vectors. For example, the standard basis for $\mathbb{R}^3$ is $\{[100],[010],[001]\}$, and any vector in $\mathbb{R}^3$ can be written as a linear combination of these 3 vectors. To illustrate this, notice that the vector $[2,-3,\pi]$ can be written as $2[100]-3[010]+\pi[001]$. There are infinitely many bases for $\mathbb{R}^3$. To find an alternate basis, we need to find 3 linearly independent vectors such that their linear combinations produce all possible vectors in $\mathbb{R}^3$.

Question 8. Produce an alternate example for a basis for $\mathbb{R}^3$.

Similarly, the standard basis for $\mathbb{R}^2$ is $\{[10],[01]\}$. In the vector space $V(4,2)$, consisting of the 16 binary strings of length 4, the standard basis is $\{[1000],[0100],[0010],[0001]\}$, since any of the other binary strings can be obtained as binary linear combinations of these vectors.

Definition 2.2.11. The *dimension* of a vector space V is number of vectors in any basis for V.

We now understand precisely why $\mathbb{R}^2$ and $\mathbb{R}^3$ are said to be 2-dimensional and 3-dimensional vector spaces over the real numbers, respectively. Moreover, $V(4,2)$ is a 4-dimensional vector space over the binary numbers.

An important concept for error-control coding is that of vector *subspaces*. Familiar subspaces from geometry are planes and lines in 3-space: Planes are 2-dimensional vector subspaces of $\mathbb{R}^3$, and lines are 1-dimensional vector subspaces of $\mathbb{R}^3$.

Definition 2.2.12. Let V be a vector space over a field F and let W be a subset of V. If W is a vector space over F, then W is a *vector subspace* of V.

Theorem 2.2.13. If W is a subset of a vector space V over a field F, then W is a vector subspace of V iff $\alpha\mathbf{u}+\beta\mathbf{v}\in W$ for all $\alpha,\beta\in F$ and all $\mathbf{u},\mathbf{v}\in W$.

Proof. Suppose that W is a vector subspace of V, a vector space over the field F. Then, W is itself a vector space over F. Therefore, by the definition of vector spaces, $\mathbf{v}+\mathbf{u}\in W$ for any $\mathbf{v},\mathbf{u}\in W$ and $s\mathbf{v}\in W$ for any $s\in F$. This directly implies that $\alpha\mathbf{u}+\beta\mathbf{v}\in W$ for all $\alpha,\beta\in F$ and all $\mathbf{u},\mathbf{v}\in W$.

Now suppose that $\alpha\mathbf{u}+\beta\mathbf{v}\in W$ for all $\alpha,\beta\in F$ and all $\mathbf{u},\mathbf{v}\in W$. Then, setting $\alpha=\beta=1\in F$ gives the first requirement of a vector space and setting $\alpha=1\in F$ and $\beta=0\in F$ gives the second requirement of a vector space. □

We can now begin to connect vector spaces to error-control codes. In Section 2.1, we discussed the linear code $C=\{000,100,001,101\}$. We will now show that this code is a vector subspace of the vector space $V(3,2)$ of binary 3-tuples. Using Theorem 2.2.13, we must check if every linear combination of vectors in C remains in C. This should sound familiar! In fact, this was the defining condition for a linear code! We have just discovered one of the major links between coding theory and linear algebra: Linear codes are vector subspaces of vector spaces.

Definition 2.2.14. A *linear code* of length n over $GF(q)$ is a subspace of the vector space $GF(q)^n = V(n,q)$.

Theorem 2.2.15 shows the consistency between the definitions for binary and general linear codes.

Theorem 2.2.15. Let C be a linear code. The linear combination of any set of codewords in C is a code word in C.

Proof. We must show that (1) $\mathbf{u} + \mathbf{v} \in C$ for all $\mathbf{u}$, $\mathbf{v}$ in C, and (2) $\alpha\mathbf{u} \in C$, for all $\mathbf{u} \in C$ and $\alpha \in GF(q)$. Since C is a subspace of $V(n, q)$, this follows from Theorem 2.2.13. □

We can deepen this connection between linear codes and vector subspaces by looking at the dimension of vector subspaces.

Example 2.2.16. Let's continue working with our linear code $C = \{000, 100, 001, 101\}$. We can see that this code is spanned by the linearly independent vectors [100] and [001], since all other vectors in C can be obtained as binary linear combinations of these two vectors. In other words, these two vectors form a basis for C, so the dimension of C is 2.

Definition 2.2.17. The *dimension* of a linear code C of length n is its dimension as a vector subspace of $GF(2)^n = V(n, 2)$.

If C is a k-dimensional subspace of $V(n, q)$, we say that C has dimension k and is an $[n, k, d]$ or $[n, k]$ linear code. (Be careful not to confuse the notation $[n, k, d]$ with our earlier notation (n, M, d), where M represents the total number of codewords in the code.) The dimension of a code is very important because it is directly related to M, the number of codewords in a code. In particular, if the dimension of a binary code is k, then the number of codewords is 2^k. In general, if the dimension of a q-ary code is k, then the number of codewords is q^k. 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 $GF(q)$. This is consistent with the aforementioned notion of encoding messages of length k to obtain codewords of length n, where $n > k$.

We can now also fully understand the output that Gap gives when defining a code. Recall the command and output

```
gap> C := RepetitionCode(5, GF(2));

a cyclic [5,1,5]2 repetition code over GF(2)
```

We now see that [5,1,5]2 means that the binary repetition code of length 5 is a [5, 1, 5] linear code. The dimension is 1 since the code is spanned by the one basis vector [11111]. The 2 after this notation refers to the covering radius of the code. We will not be covering this concept in this course.

2.3 Encoding Linear Codes: Generator Matrices

Linear codes are used in practice largely due to the simple encoding procedures facilitated by their linearity. A $k \times n$ *generator matrix* G for an $[n, k]$ linear code C provides a compact way to describe all of the codewords in C and provides a way to encode messages. By performing the multiplication $\mathbf{m}G$, a generator matrix maps a length k message string $\mathbf{m}$ to a length n codeword string. The encoding function $\mathbf{m} \to \mathbf{m}G$ maps the vector space $V(k, q)$ onto a k-dimensional subspace (namely the code C) of the vector space of $V(n, q)$.

Before giving a formal definition of generator matrices, let's start with some examples.

Example 2.3.1. We will use the familiar repetition code example to illustrate how we can use matrices to encode linear codes. As you know, in the repetition code, 0 is encoded as 00000 and 1 is encoded as 11111. This process can be automated by using a generator matrix G to encode each message: If $G = [11111]$, then for a message $\mathbf{m}$, the matrix multiplication $\mathbf{m}G$ outputs the

corresponding codeword. Namely $[0]G = [00000]$ and $[1]G = [11111]$. So, this 1×5 matrix G is working to map a message of length 1 to a codeword of length 5. Since we knew exactly how our two messages encode to our two codewords, it was very easy to find this matrix. ✓

Example 2.3.2. We will now extend the work done in Example 2.2.16 on $C = \{000, 100, 001, 101\}$. We have already shown that C is a 2-dimensional vector subspace of the vector space $V(3, 2)$. Therefore, C is a $[3, 2]$ linear code and should have a 2×3 generator matrix. However, let's use our intuition to derive the size and content of the generator matrix. We want to find a generator matrix G for C that will map messages (*i.e.* binary bit strings) onto the four codewords of C. But how long are are the messages that we can encode? We can answer this question using three facts: (1) we always encode messages of a fixed length k to codewords of a fixed length n; (2) a code must be able to encode arbitrary messages of length k; and (3) the encoding procedure must be "one-to-one," or in other words, no two messages can be encoded as the same codeword. Using these three facts, it is clear that the messages corresponding to the four codewords in C must be of length 2 since there are exactly 4 distinct binary messages of length 2: [00], [10], [01], and [11]. Hence, in order for the matrix multiplication to make sense, G must have 2 rows. Since the codewords are of length 3, properties of matrix multiplication tell us that G should have 3 columns. In particular, if $G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, then $[00]G = [000]$, $[10]G = [100]$, $[01]G = [001]$, and $[11]G = [101]$. Hence, this code C can encode any length 2 binary message. The generator matrix determines which length 2 message is encoded as which length 3 codeword. ✓

We have now seen how a generator matrix can be used to map messages onto codewords. We have also seen that an $[n, k]$ linear code C is a k-dimensional vector subspace of $V(n, q)$ and can hence be specified by a basis of k codewords, or q-ary vectors. This leads us to the following formal definition of a generator matrix:

Definition 2.3.3. A $k \times n$ matrix G whose rows form a basis for an $[n, k]$ linear code C is called a *generator matrix* of the code C.

Definition 2.3.3 shows that the dimension of a code C is equal to the number of rows in its generator matrix, which is consistent with our above examples. In Example 2.3.1, we found a 1×5 generator for our [5,1] repetition code (length 5 and dimension 1, since it is spanned by the vector [11111]). In Example 2.3.2, we indeed found a 2×3 generator matrix for the [3, 2] linear code C.

Since Definition 2.3.3 stipulates that the rows of a generator matrix must form a basis for the code, the rows of a generator matrix must be linearly independent. 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.

Example 2.3.4. The following matrix is not a generator matrix for any linear code:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

To see this, notice that the rows cannot form a basis for a linear code because the rows are linearly dependent. There are several ways to discover this. For example, you can see that (bit-wise binary) adding the first and second rows gives you the fourth row or reveal the dependency by obtaining a row of zeros through row-reduction. Alternately, you can note that the messages [1100] and [0001] are both encoded to the same codeword [01110]. Similarly, the messages [1101] and [0000] both encode to [00000]. ✓

Without a generator matrix to describe the code and define the encoding of messages, it would be necessary to make a list of all of the codewords and their corresponding messages. This is very simple for the repetition code, however, this would become highly inconvenient as the number of codewords (and therefore messages) increases.

Question 9. Given the generator matrix $G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$, what is the dimension of the associated code? How long are the messages to be encoded by G? How long will the resulting codewords be? Use G to encode the all-1s message of the appropriate length.

We can use Gap to facilitate the generation of linear codes from their matrices, and the subsequent encoding of messages. The command `GeneratorMatCode` generates a code based on a generator matrix. For example, suppose our generator matrix of a binary code is:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Then, the basis vectors of this code are `[1 0 0 0 1]`, `[0 1 0 1 0]`, and `[0 0 1 1 1]`. We can now use Gap to construct our linear code from this generator matrix (or basis for the code):

```
gap> C := GeneratorMatCode([[1,0,0,0,1],[0,1,0,1,0],[0,0,1,1,1]],GF(2));;
```

To review the generator matrix for this code C, we can use a combination of the functions `Display` and `GeneratorMat`. For example:

```
gap> Display(GeneratorMat(C));
 1 . . . 1
 . 1 . 1 .
 . . 1 1 1
```

The dots in the output represent 0's.

We can also use Gap to encode messages. For example, to encode `[1, 1, 1]` we use:

```
gap> m := Codeword("111", GF(2));
gap> m * C;
[ 1 1 1 0 0 ]
```

Notice we do not have to use `GeneratorMat` to reproduce the generator matrix to encode the message `m`. We can encode the message directly by multiplying the message, `m` with the code, `C`.

Question 10. Confirm the above encoding by hand.

We have now seen how generator matrices can be used to encode messages. Generator matrices are also used to discover codes, as illustrated in the following example:

Example 2.3.5. Suppose we need a [3,2] binary linear code C, that is, we need a 2-dimensional vector subspace of $V(3,2)$, the vector space of binary 3-tuples. In order to define a 2-dimensional vector subspace of $V(3,2)$, we need 2 linearly independent basis vectors. Arbitrarily, let's choose these vectors to be 011 and 110. Let C be spanned by these vectors. Then the generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Taking all the linear combinations of the rows in G, we generate the code $C = \{110, 011, 101, 000\}$.✓

Question 11. Use matrix multiplication to convince yourself that taking all linear combinations of the rows of the 2×3 matrix G above is equivalent to taking all products $\mathbf{m}G$, where m runs through all 4 binary 2-tuples. In particular, show that the product $[11]G$ produces a vector which is the sum of all rows in G.

There was nothing special about the basis vectors [110] and [011] that we chose in the above example. In fact, there would be advantages to choosing permutations of these vectors obtained by swapping the first two entries in each vector. This would produce the generator matrix

$$G' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

This is the *systematic or standard form* of G since it has the identity matrix on the right of G. When a data block is encoded using a generator matrix in standard form, the data block is embedded without modification in the last k coordinates of the resulting codeword. This facilitates decoding, especially when the decoding is implemented with linear feedback shift registers. When data is encoded in this fashion, it is called *systematic encoding*.

Question 12. Using G' above, encode all possible length 2 messages to confirm that the messages appear in the last 2 bits of each codeword.

We can go one step further into the theory of linear algebra to deepen the understanding of the connections between codes and matrices.

Definition 2.3.6. The *rowspace* of an matrix is the set of all vectors that can be obtained by taking linear combinations of the rows of the matrix.

Recall that taking all linear combinations of the rows of an $r \times c$ matrix G is equivalent to taking products $\mathbf{v}G$, where $\mathbf{v}$ is any $1 \times r$ vector. It follows that the code C represented by a generator matrix G is equivalent to the rowspace of G.

Example 2.3.7. The rowspace of the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

is all of $\mathbb{R}^3$, since any real vector $[xyz]$ in $\mathbb{R}^3$ can be formed as a real linear combination of the rows of M. More precisely, any real vector $[xyz] \in \mathbb{R}^3$ can be written as $[xyz]M$, which represents a linear combination of the rows of M.

Question 13. Find a 3×3 matrix whose rowspace represents a plane in $\mathbb{R}^3$. Recall that a plane is a 2-dimensional subspace of $\mathbb{R}^3$.

Question 14. Is the matrix you defined in the previous question a generator matrix for a linear code?

We have just introduced several equivalent constructs: vector subspaces, linear codes, and row spaces of matrices. As we delve further into the intriguing topics of coding theory, linear algebra, and abstract algebra, we will see that the lines between pure mathematics and real-world applications continues to blur.

2.4 Introduction to Parity Check Matrices

Definition 2.4.1. Let C be an $[n, k]$ linear 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 $\mathbf{c}H^T = 0$.

Example 2.4.2. If C is the code $\{111, 000\}$, then a valid parity check matrix is

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

One way to verify this is to compute the product $\mathbf{c}H^T$ for an arbitrary codeword $\mathbf{c} = [x, y, z]$:

$$[xyz] \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = [x + z, y + z]$$

So, $c = [xyz]$ is a valid codeword iff $cH^T = 0$, ie. iff $[y + z, x + z] = [0, 0]$. Algebraic manipulations over binary show that $x = y = z$. Thus the only valid codewords are indeed $\{000, 111\}$. Notice that this is the binary repetition code of length 3. ✓

Question 15. Find a parity check matrix for the binary repetition code of length 5.

Example 2.4.3. The (10,9) binary parity check code (BPCC) is a code of length 10 and dimension 9 which encodes binary messages of length 9 into binary codewords of length 10 by appending an overall parity check bit. For example, the message [001100100] is encoded as [0011001001] and the message [100101100] is encoded as [1001011000]. The parity check matrix for this code is $H = [1111111111]$ since then the product $\mathbf{c}H^T$ forms the sum (modulo 2) of the bits in the length 10 codeword $\mathbf{c}$; this product gives 0 precisely when the parity of the codeword is even, or in other words, precisely $\mathbf{c}$ is a member of the BPCC. ✓

Question 16. Find a parity check matrix for the (5, 4) binary parity check code, which encodes messages of length 4 to codewords of length 5 by appending a parity check bit.

The Guava function `CheckMat` generates a parity check matrix for any given linear code. For example, we can generate a check matrix for a repetition code of length 5 as follows:

```
gap> C := RepetitionCode(5, GF(2));;

gap> H := CheckMat(C);;

gap>Display(H);
 1 1 . . .
 . 1 1 . .
 . . 1 1 .
 . . . 1 1
```

Question 17. Show that any codeword $\mathbf{c}$ in the binary repetition code of length 5 indeed satisfies the equation $\mathbf{c}H^T$, where H is given by the Gap output above.

The connection drawn above between the codewords of a code C and the parity check matrix of a code C can be strengthened by considering the nullspace of the parity check matrix.

Definition 2.4.4. The *nullspace* of a matrix H is the set of all vectors $\mathbf{x}$ such that $H\mathbf{x}^T = 0$.

In linear algebra, we study nullspaces of matrices, often in the context of finding the *kernel* of linear mappings. However, there is a simple connection to linear codes. Given the parity check matrix H for a code C, we have $0 = \mathbf{c}H^T$ for all $\mathbf{c} \in C$, hence $0^T = (\mathbf{c}H^T)^T = H\mathbf{c}^T$. Therefore, the codewords of C comprise the nullspace of its parity check matrix H.

Question 18. Find the nullspace of

$$B = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Theorem 2.4.5. An $(n-k) \times n$ parity check matrix H for a code C in *systematic* or *standard form* if $H = (I_{n-k} \mid A)$. Then the corresponding generating matrix in *systematic form* is $G = (-A^T \mid I_k)$.

Notes: (1) Authors vary in their definitions of the systematic form of generator and parity check matrices; some put the identity matrix to the left for systematic generator matrices and to the right for systematic parity check matrices; and (2) When working in binary, the negative sign in $G = (-A^T \mid I_k)$ is moot.

Question 19. Find systematic generator matrices for the codes defined by the parity check matrices in Examples 2.4.2 and 2.4.3. Note that these parity check matrices are given in systematic form.

Any parity check matrix or generating matrix can be put in systematic form by performing row operations on the matrix or column permutations. This substantially 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. Depending on the operations used to convert the defining matrices, either the same code or an *equivalent* code will be obtained.

Definition 2.4.6. 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 $GF(q)$).

Example 2.4.7. The two codes C_1 and C_2 below are equivalent since the second is merely the permutation of the first two positions in the first code.

$$C_1 = \left\{ \begin{array}{l} 000 \\ 101 \\ 010 \\ 111 \end{array} \right\} \quad C_2 = \left\{ \begin{array}{l} 000 \\ 011 \\ 100 \\ 111 \end{array} \right\}$$

Theorem 2.4.8. Two $k \times n$ matrices generate equivalent $[n, k]$ linear codes over $GF(q)$ 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. □

Gap can be used to put parity check matrices and generator matrices into systematic or standard form. It is necessary to remember which standard form requires the identity matrix on the left. For our convention, we will put the identity on the right in generator matrices and on the left for parity check matrices.

For example, suppose I have this check parity matrix

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

We can represent this check matrix in a systematic form by using the `PutStandardForm` command which inputs two values, the matrix and a boolean variable, true or false. `PutStandardForm(H, true)` will arrange the input matrix H so that the identity matrix is on the left. `PutStandardForm(H, false)` will arrange the input matrix H so that the identity matrix is on the right.

```
gap> H := [[1,0,1,0], [0, 1, 1, 0], [0, 0, 0, 1]];

gap> Display(H);
[ [  1,  0,  1,  0 ],
  [  0,  1,  1,  0 ],
  [  0,  0,  0,  1 ] ]

gap> PutStandardForm(H, true);;

gap> Display(H);
[ [  1, 0,  0,  1 ],
  [  0, 1,  0,  1 ],
  [  0, 0,  1,  0 ] ]
```

Note that the (3,4) output indicates that the arranged matrix is 3×4, and then we use the `Display` command to view the standard form of H.

2.5 Parity Check Matrices and Linear Decoding

This section discusses a nearest neighbor decoding scheme that uses parity check matrices and cosets (which you first encountered when cryptanalyzing the Vigenere cipher) to decode a linear code.

The defining equation of a parity check matrix ($c \in C \Leftrightarrow cH^T = 0$) is used in the first step of the decoding of linear codes. Upon receiving a word $\mathbf{r}$, the product $\mathbf{r}H^T$ is computed. If the result is 0, we assume that no errors have occurred and decode the received word as $\mathbf{r}$.

Question 20. Is this a fair assumption?

We will now see how to decode if $\mathbf{r}H^T \neq 0$.

Definition 2.5.1. Suppose that C is an $[n, k]$ linear code over $GF(q)$ 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 2.5.2. Suppose C is an $[n, k]$ linear code over $GF(q)$. 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 (non-overlapping) or equal.

Example 2.5.3. Let C be the binary $[4, 2]$ linear code with generator matrix $G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$. 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.✓

Definition 2.5.4. The *coset leader* of a coset is chosen to be one of the vectors of minimum weight in the coset.

Example 2.5.5. In Example 2.5.3, 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 $GF(q)^n$ 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 $\mathbf{r}$ is received, find the coset that contains $\mathbf{r}$. This coset is of the form $\mathbf{e} + C$, and we guess that $\mathbf{r} - \mathbf{e}$ 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 $GF(q)^n$ 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 2.5.3 is:

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

We decode $\mathbf{r}$ as the codeword at the top of the column containing $\mathbf{r}$. 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 cannot 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 21. Explain why the above code, with the above decoding scheme, cannot correct errors in the fourth position.

Question 22. Construct a standard array for the binary repetition code of length 4.

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 $\mathbf{r}$ is received. Compute $\mathbf{r}H^T$, which we call the *syndrome* of $\mathbf{r}$ and write $S(\mathbf{r})$. We know from the definition of a parity check matrix that $\mathbf{r}H^T = \mathbf{0}$ if and only if $\mathbf{r} \in C$. So, if $S(\mathbf{r}) = \mathbf{r}H^T = \mathbf{0}$, 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 cannot detect or correct this.) If $\mathbf{r}H^T \neq \mathbf{0}$, then we know that at least one error occurred, and so $\mathbf{r} = \mathbf{x} + \mathbf{e}$, where $\mathbf{x}$ is a codeword and $\mathbf{e}$ is an error vector. Since $\mathbf{x}H^T = 0$, we see that $\mathbf{r}H^T = (\mathbf{x} + \mathbf{e})H^T = \mathbf{x}H^T + \mathbf{e}H^T = \mathbf{e}H^T$. So the syndrome of the received word $\mathbf{r}$ 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 2.5.6. When the standard array of Example 2.5.3 is expanded to allow for syndrome

decoding, it looks as follows:

$$\begin{array}{ccccc} 0000 & 1011 & 0101 & 1110 & 00 \\ 1000 & 0011 & 1101 & 0110 & 11 \\ 0100 & 1111 & 0001 & 1010 & 01 \\ 0010 & 1001 & 0111 & 1100 & 10 \end{array} .\checkmark$$

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

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 2.5.7. Suppose that we are decoding using the expanded standard array shown in Example 2.5.6. 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 $1111 - 0100 = 1011$. ✓

The following theorem will help connect the ideas of syndromes and extended arrays to parity check matrices:

Theorem 2.5.8. For a binary code, the syndrome of a received codeword is a vector that 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

$$\begin{aligned} S(\mathbf{r}) &= \mathbf{r}H^T \\ &= (\mathbf{c} + \mathbf{e})H^T \\ &= \mathbf{c}H^T + \mathbf{e}H^T \\ &= \mathbf{0} + \mathbf{e}H^T \qquad (6) \\ &= \mathbf{e}H^T \end{aligned}$$

Notice that (6) 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. □

Decoding linear codes, calculating syndromes, and computing standard and extended arrays can be done using Gap. If all we want to know is whether a codeword is in our code or not, we can use the `in` operator. For example:

```
gap> c := Codeword("10111", GF(2));;

gap> C := RepetitionCode(5,GF(2));;

gap> c in C;
false
```

We can also use the built-in function for calculating syndromes. We illustrate this with an example of computing the syndrome of the codeword [11111] in the binary repetition code of length 5:

```
gap> C := RepetitionCode(5, GF(2));;

gap> c := Codeword("11111", GF(2));;

gap> s := Syndrome(C,c);
[ 0 0 0 0 ]
```

The function `StandardArray` is a function that will return the standard array of a linear code C. For example, we can construct a standard array for a repetition code of length 4.

```
gap> StandardArray(RepetitionCode(4, GF(2)));
[ [ [ 0 0 0 0 ], [ 1 1 1 1 ] ], [ [ 0 0 0 1 ], [ 1 1 1 0 ] ],
  [ [ 0 0 1 0 ], [ 1 1 0 1 ] ], [ [ 0 0 1 1 ], [ 1 1 0 0 ] ],
  [ [ 0 1 0 0 ], [ 1 0 1 1 ] ], [ [ 0 1 0 1 ], [ 1 0 1 0 ] ],
  [ [ 0 1 1 0 ], [ 1 0 0 1 ] ], [ [ 1 0 0 0 ], [ 0 1 1 1 ] ] ]
```

Be sure to look at this output carefully to notice the placement of brackets. If we were to write out this standard array by hand, we would write eight rows with two entries each.

The function `SyndromeTable` is a function that will return an abbreviated version of our extended standard array. The first column in the output of `SyndromeTable` consists of the error vectors that correspond to the syndrome vectors in the second column. So, the output of `SyndromeTable` contains only the first and last columns of our extended standard arrays. To make this clear, we will now produce the syndrome table for the same above repetition code:

```
gap> SyndromeTable(RepetitionCode(4, GF(2)));
[ [ [ 0 0 0 0 ], [ 0 0 0 ] ], [ [ 0 0 0 1 ], [ 0 0 1 ] ],
  [ [ 0 0 1 1 ], [ 0 1 0 ] ], [ [ 0 0 1 0 ], [ 0 1 1 ] ],
  [ [ 1 0 0 0 ], [ 1 0 0 ] ], [ [ 0 1 1 0 ], [ 1 0 1 ] ],
  [ [ 0 1 0 0 ], [ 1 1 0 ] ], [ [ 0 1 0 1 ], [ 1 1 1 ] ] ]
```

You will notice that the output of `SyndromeTable` is sorted according to the syndrome vectors. Although this may be confusing when you compare the output of `SyndromeTable(RepetitionCode(4, GF(2)));` to the output of `StandardArray(RepetitionCode(4, GF(2)));`, Gap does this to facilitate the search of the syndrome vectors, which is of utmost importance in decoding linear codes.

2.6 Parity Check Matrices, Minimum Distance, and the Singleton Bound

In addition to contributing to the decoding scheme for linear codes, parity check matrices can be used to determine the minimum distance of the code:

Theorem 2.6.1. 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:

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

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

Theorem 2.6.1 can be used to bound the possible minimum distance for a code, as explained in the following theorem.

Theorem 2.6.2. *(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. Linear algebra results then imply that 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 2.6.1. □

Codes that achieve the Singleton bound with equality are called *maximum distance separable* (MDS). For their given parameters n and k, they achieve the best possible minimum distance.

Question 24. Show that the binary repetition code of length 5 is MDS.

Gap can be used to check if a code is MDS. The following commands and output confirm that the binary repetition code of length 5 is MDS:

```
gap> C := RepetitionCode(5, GF(2));;

gap> IsMDSCode(C);

true
```

In Chapter 4.3, we will study some more interesting MDS codes.

2.7 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 2.7.1. 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 25. 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:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

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

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

A generator matrix G in standard form is:

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

Question 26. Encode the messages 0000 and 1010 using G. Check that the resulting codewords are valid by using H.

Guava has built-in functions for Hamming codes. The $[7, 4, 3]$ binary Hamming code is defined by the parameter $r = 3$, and is generated as follows:

```
gap> C := HammingCode(3, GF(2));

a linear [7,4,3]1 Hamming (3,2) code over GF(2)
```

To view the full list of codewords in the $[7, 4, 3]$ binary Hamming code, we use the following command:

```
gap> cw := AsSortedList(C);
    [[0000000], [0001111], [0010110], [0011001],
     [0100101], [0101010], [0110011], [0111100],
     [1000011], [1001100], [1010101], [1011010],
     [1100110], [1101001], [1110000], [1111111]]
```

Notice that the output is sorted. The list returned is also immutable, meaning that elements of the list cannot be changed. It will sometimes be useful to be able to access the elements in the outputted list. For example, in what follows, we will add together two of the codewords in the list and check that their sum is also a codeword (guaranteed by the linearity of the code):

```
gap> d := cw[2] + cw[3];
[ 0 0 1 1 0 0 1 ]

gap> d in cw;
true
```

Question 27. Write down a parity check matrix for the binary Hamming code with $r = 4$ by using the mathematical definition of Hamming codes. Check your answer with Gap.

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 $\mathbf{r}$, compute the syndrome $S(\mathbf{r})$. If $S(\mathbf{r}) = \mathbf{0}$, then assume that $\mathbf{r}$ was the codeword sent. If $S(\mathbf{r}) \neq \mathbf{0}$, then, assuming a single error, $S(\mathbf{r})$ is equal to the column of H that corresponds to the coordinate of $\mathbf{r}$ where the error occurred. Find the column of H that matches $S(\mathbf{r})$, and then correct the corresponding coordinate of $\mathbf{r}$. This decoding scheme is further simplified if the columns of H are arranged in order of increasing binary numbers.

Example 2.7.2. In Section 1.3, 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:

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

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.✓

2.8 Answers to Chapter 2 Reading Questions

Answer 1. This Hamming code can correct 1 error.

Answer 2. The first code is not linear because it does not contain 101, the sum of 100 and 001. The second code is not because it does not contain 000 (which is the binary sum of any codeword with itself).

Answer 3.
```
gap> C := ElementsCode(["000", "100", "001"], GF(2));; gap>
IsLinearCode(C): false

gap> C := ElementsCode(["100", "001", "101"], GF(2));; gap>
IsLinearCode(C): false
```

Answer 4. Let C be a binary linear code. Let $\mathbf{c}$ be a codeword in C. Since C is linear, $(\mathbf{c} + \mathbf{c}) = \mathbf{0}$ must also be in C.

Answer 5. We would need to consider $\binom{M}{2}$ pairs of codewords to find the minimum distance of a code with M codewords.

Answer 6. The additive inverse of 0 is 0. The additive inverse of 1 is 6 and its multiplicative inverse is 1. The additive inverse of 2 is 5, and its multiplicative inverse is 4. The additive inverse of 3 is 4, and its multiplicative inverse is 5. The additive inverse of 4 is 3, adn its multiplicative inverse is 2. The additive inverse of 5 is 2, and its multiplicative inverse is 3. The additive inverse of 6 is 1, and its multiplicative inverse is 6.

Answer 7. $\{[1, 1, 1], [0, 3, 4], [3, 9, 11]\}$ are linearly dependent since, for example, $-3[1, 1, 1]+2[0, 3, 4]+[3, 9, 11] = [0, 0, 0]$. $\{[1, 0, 1], [12, 3, 0], [13, 3, 0]\}$ are linearly independent.

Answer 8. $\{[2, 0, 0], [0, 4, 0], [0, 0, 5]\}$ is an alternate basis.

Answer 9. The associated code has dimension 3 since there are three (linearly independent) rows in G and thus three basis vectors. The messages that are to be encoded by G are of length 3. The resulting codewords have length 4, which is the number of columns in G. The all-1s message will be encoded as [1111].

Answer 10. $(111)G = (11100)$

Answer 11. The four binary 2-tuples are $\{[00], [01], [10], [11]\}$. When performing matrix multiplication, notice that the above four binary 2-tuples designate all the different ways that we can add rows together. For example, we can see that $[11]G$ is equivalent to adding all the rows of G, resulting in $[110] + [011] = [101]$.

Answer 12. $[00]G' = [000]$, $[01]G' = [101]$, $[10]G' = [110]$, $[11]G' = [011]$

Answer 13. $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

Answer 14. No, this matrix is not a generator matrix for a linear code. Its rows are not linearly independent so they do not form a basis for a linear code.

Answer 15. We know that H is an $(n-k)\times n$ matrix by definition, so the first step is to determine n and k. We know that the dimension of any repetition code is 1 since it contains only two codewords: a string of ones and a string of zeroes. We also know that $n = 5$ since the length was given in the problem statement. So we must determine a 4×5 matrix such that the product of 11111 or 00000 with H^T yields the zero vector. 00000 will work with any matrix, so we can craft the matrix based on what 11111 must multiply with to get the zero vector. This works when every *column* of H^T has an even number of ones, or, when every *row* of H has an even number of ones. One way of creating this matrix is shown below.

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Answer 16. Since every codeword has an even number of ones, any codeword multiplied by the all-ones vector will yield zero, as in Example 2.4.3.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Answer 17. $(00000)H^T = (0000)$
$(11111)H^T = (0000)$

Answer 18. We know that the nullspace consists of every vector $\mathbf{x}$ such that $H\mathbf{x}^T = 0$. So we have

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Then,

$$x_1 + x_2 = 0$$
$$x_1 + x_3 = 0.$$

By algebra, $x_1 = x_2 = x_3$, so the nullspace of B is $000, 111$.

Answer 19. $G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Answer 20. Yes, this is a fair assumption. It is true unless so many errors have occurred that we started with one valid codeword and ended up with a different valid codeword. In that case, there would be no way to detect that anything went wrong and the received vector would be decoded incorrectly.

Answer 21. There are a total of 16 binary length 4 vectors, and this code contains 4 of these vectors. So our standard array can only have 4 rows. Thus, only three weight 1 vectors can be coset leaders. In this case, (0001) is not a coset leader, so a single error in this position will not be decoded correctly.

Answer 22.

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

By examining the coset leaders, we can see that this corrects all single-errors but only three types of double-errors.

Answer 23. No single column is zero, so the minimum distance is not 1. No two columns have a linear combination that sums to 0, so the minimum distance is not 2. However, the third column is a linear combination of the first two, so the minimum distance of C is 3.

Answer 24. Since the binary repetition code of length 5 satisfies the Singleton bound with equality, it is MDS. Check this using $d = 5$, $n = 5$, and $k = 1$.

Answer 25. A Hamming code can correct one error since it has a distance of 3.

Answer 26. $(0000)G = (0000000)$
$(1010)G = (1011010)$
$(0000000)H^T = (000)$
$(1011010)H^T = (000)$

Answer 27. We can form this matrix by writing down as columns all of the length 4 binary vectors except for the zero vector. We put the identity on the left side in order for the parity check matrix to be in standard form. $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

2.9 Chapter 2 Problems

Problem 2.1. Are the following sets of vectors linearly independent?

1. $[101111], [000001], [110111] \in \mathbb{Z}_2^6$.
2. $[000], [101] \in \mathbb{Z}_2^3$.
3. $[10000], [01001], [00111] \in \mathbb{Z}_2^5$.
4. $[0212], [0010], [2212] \in \mathbb{Z}_3^4$.

Problem 2.2. Let $G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$. Determine whether or not the following words are in the rowspace of G:

1. [11111]
2. [11101]
3. [00111]

Problem 2.3. Let $C = \{[000000], [101110], [001010], [110111], [100100], [011001], [111101], [010011]\} \subseteq \mathbb{Z}_2^6$. Find a generator matrix for C and a parity check matrix for C.

Problem 2.4. List the cosets of the linear code having the following generator matrix:

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

Problem 2.5. Construct a standard array for a binary code having the following generator matrix: $\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$. 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.

Problem 2.6. Construct a syndrome look-up table for the perfect binary $[7, 4, 3]$ code which has generator matrix $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$. Use the table to decode the following received vectors: 0000011, 1111111, 1100110, and 1010101. Repeat this exercise using Gap.

Problem 2.7. Let $C \subseteq \mathbb{Z}_2^6$ be the linear $[6, 3]$ code with generator matrix. $G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

1. List the elements of C.
2. Find the minimum distance of C.
3. How many errors will C correct?
4. Find a check matrix for C.

Problem 2.8. Let C be a q-ary linear code with parity check matrix H. Let $\mathbf{u}, \mathbf{v} \in GF(q)^n$. Prove directly that $\mathbf{u}H^T = \mathbf{v}H^T$ if and only if $\mathbf{u}$ and $\mathbf{v}$ lie in the same coset of C.

Problem 2.9. Define an encoding map E: $\mathbb{Z}_2^4 \to \mathbb{Z}_2^6$ by $E((x_1, x_2, x_3, x_4)) = (x_1, x_2, x_2, x_3, x_4, x_4)$. Show that the encoding map E represents the encoding of a linear code. In other words, show that the codewords produced under this map satisfy the requirements of a linear code.

Problem 2.10. Define an encoding map E: $\mathbb{Z}_2^6 \to \mathbb{Z}_2^6$ by $E((x_1, x_2, x_3, x_4, x_5, x_6)) = (x_3, x_4, x_1, x_5, x_2, x_6)$. Show that the encoding map E represents the encoding of a linear code. In other words, show that the codewords produced under this map satisfy the requirements of a linear code.

Problem 2.11. Let C be a linear $[n, k]$ code over $\mathbb{Z}_p$, where p is a prime number. How many elements does C contain? Explain your answer.

Problem 2.12. Let C be a ternary linear code with generator matrix $\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$. List the codewords of C. What is the minimum distance of C? Is C a perfect code?

Problem 2.13. Extended Codes:

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. 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.)

3. Generalize the idea of extending binary codes with an overall parity check matrix to define extended codes of q-ary codes. What condition should the weight of each codeword in the extended code satisfy?

Problem 2.14. Prove that either all of the codewords in a binary linear code have even weight or exactly half have even weight.

Problem 2.15. Let $U(n) \subseteq \mathbb{Z}_n$ be the set of elements of $\mathbb{Z}_n$, that are relatively prime to n. Show that $U(n)$ is a group under multiplication modulo n.

Problem 2.16. Prove that the identity element in a group is unique

Problem 2.17. Prove that if G is a group and $a \in G$, then a^{-1} is unique.

Problem 2.18. 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.

Problem 2.19. Prove that any Hamming code has minimum distance equal to 3.

Problem 2.20. Prove that binary Hamming codes are perfect.

Problem 2.21. 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 $GF(q)$.

Problem 2.22. Let $A = \{0, 1\}$.

1. Does there exist a code $C \subseteq A^8$ where C has exactly four codewords and such that C corrects 2 errors? If so, find such a C. If not, prove it

2. Does there exist a code $C \subseteq A^8$ where C has exactly five codewords and such that C corrects 2 errors? If so, find such a C. If not, prove it.

Problem 2.23. Alice and Bob have used the Bennett-Brassard protocol to generate two strings of bits which ideally are identical but which in fact contain some discrepancies. They have determined that it is very unlikely that a block of seven bits will contain more than one error; so they have decided to use the [7,4] binary Hamming code to correct their errors.

1. Bob's first block of seven bits is 1110110. Alice sends him the syndrome 111. What does Bob's first block become after error correction?

2. Bob's second block is 0000111. Alice sends him the syndrome 111. What does Bob's second block become after error correction?

Problem 2.24. Prove that any linear ternary code (i.e. its alphabet has three symbols, 0, 1, 2 under modulo 3) with a check matrix H can correct all single errors if and only if no two columns of H have sum or difference 0.

3 Cyclic Codes, Rings, and Ideals

This chapter studies more advanced topics from both abstract algebra and coding theory.

3.1 Introduction to Cyclic Codes

Definition 3.1.1. 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 3.1.2. The binary code $C = \{000, 110, 011, 101\}$ 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 $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ of length n 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.

Gap can be used to convert a codeword to a codeword polynomial. For example:

```
gap> c := Codeword("0111000", GF(2));;

gap> PolyCodeword(c);

x_1^3+x_1^2+x_1
```

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

Question 2. For a codeword of length 5, what is the maximum degree of the associated code polynomial?

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.

The power of using code polynomials will be apparent when we develop some additional structure. We will parallel the development of $\mathbb{Z}_p$, the integers modulo p, from the regular integers $\mathbb{Z}$. Recall that with the integers modulo p, we constantly wrap around according to the modulus p. In other words, the largest number we can use is $p - 1$, since as soon as we hit p or larger, we are reduced modulo p. We think of p as being equivalent to 0, so all multiples of p are "ignored."

Now consider the code polynomials $\{0, 1 + x, x + x^2, 1 + x^2\}$ corresponding to the code $C = \{000, 110, 011, 101\}$. Notice that the highest power appearing among the code polynomials is x^2, since a higher power term would indicate a codeword with length greater than 3. Similarly to working with integers modulo a specific prime integer p, we can work with polynomials modulo a specific polynomial, for example $p(x) = x^3 - 1$. In this situation, we think of $x^3 - 1$ being equivalent to 0, or in other words, x^3 is equivalent to 1. Let's see what this means with an example.

Example 3.1.3. Consider the code polynomial $c(x) = 1+x^2$ corresponding to the codeword $\mathbf{c} = 101$ from $C = \{000, 110, 011, 101\}$. Let's multiply $c(x)$ by x to obtain $c(x)x = c'(x) = x + x^3$. However, if we are working modulo $p(x) = x^3 - 1$, then x^3 is equivalent to 1. So, $c'(x)$ is equivalent to $x + 1$ modulo $p(x)$. Notice that rearranging this new polynomial with terms from lowest to highest powers gives $1 + x$, which is the code polynomial for the code word $110 \in C$. Notice furthermore that 110 is the right cyclic shift of 101. Coincidence? Let's try this again. This time, start with the code polynomial $d(x) = x + x^2$ corresponding to the codeword 011 in C. Multiplying $d(x)$ by x gives $d'(x) = x^2 + x^3$. Taking this result modulo $x^3 - 1$ gives $x^2 + 1$, which (upon rearranging terms from lowest to highest powers) corresponds to the codeword $101 \in C$ and is the right cyclic shift of 011. Interesting!✓

Example 3.1.3 suggests a true fact about cyclic codes. In a cyclic code C of length n, the product $xc(x)$ modulo $x^n - 1$ produces another code polynomial in C, namely the right cyclic shift of

$c(x)$. More precisely, when working with code polynomials of degree less than n corresponding to codewords of length n, 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 $c(x) = c_0+c_1x+\cdots+c_{n-1}x^{n-1}$. Multiplying $c(x)$ by x modulo x^n-1 gives $c'(x) = c_0x+c_1x^2+\cdots+c_{n-1}x^n \equiv c_0x + c_1x^2 + \cdots + c_{n-1}$ 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 $c(x)$.

Question 3. Show that multiplying $c(x) = 1+x^2$ by x^2 corresponds to doubly-shifting the associated codeword.

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

Theorem 3.1.4. A linear code of length n over $GF(q)$ is cyclic if and only if C satisfies the following two conditions:

1. If $a(x)$ and $b(x)$ are code polynomials in C, then $a(x) - b(x) \in C$
2. If $a(x)$ is a code polynomial in C and $r(x)$ is any polynomial of degree less than n, then $r(x)a(x) \in C$

Proof. Suppose C is a cyclic code of length n over $GF(q)$. Then C is linear, so Condition (1) holds. Now suppose that $a(x) \in C$ and $r(x) = r_0 + r_1x + \ldots r_{n-1}x^{n-1}$, where $r_i \in GF(q)$. As discussed above, multiplication of a code polynomial by x corresponds to a cyclic shift of the corresponding codeword. Since C is a cyclic code, it contains the cyclic shifts of all codewords, so $xa(x) \in C$. Similarly, $x^ic(x)$, for $0 < i < n$, is the i^{th} cyclic shift of $c(x)$, so $x^ic(x) \in C$. Now, by the linearity of C, $r(x)a(x) = r_0a(x) + r_1xa(x) + \cdots + r_{n-1}x^{n-1}a(x)$ is also in C since each summand is in C. Therefore, Condition (2) also holds. Hence, if C is a cyclic code, then Conditions (1) and (2) hold.

On the other hand, suppose that Conditions (1) and (2) hold. If we take $r(x)$ to be a scalar in $GF(q)$, 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. Hence, if Conditions (1) and (2) hold, then C is a cyclic code. □

We can use Gap to check if a given code `C` is cyclic using the `IsCyclicCode(C);` command.

3.2 Rings and Ideals

Above, in working with cyclic codes of length n, we used code polynomials of degree less than n and polynomial multiplications modulo $x^n - 1$. This set of elements (code polynomials of degree less than n) and this operation (polynomial multiplication modulo $x^n - 1$) forms what is known as a *ring*. Rings are mathematical structures that are slightly less restrictive than fields. The only difference between fields and the type of rings that we will consider in this course is that our rings do not require the existence of multiplicative inverses. The other 8 criteria from the Definition 2.2.1 of fields apply.

For completeness, we give the formal definition of a ring:

Definition 3.2.1. Let R be a nonempty set that is *closed* under two binary operations denoted $+$ and $*$. Then $(R, +, *)$ is a *ring* if:

1. There exists an *additive identity* denoted 0 in R such that $a + 0 = 0 + a = a$ for any element $a \in R$.
2. The addition is *commutative*: $a + b = b + a$, for all $a, b \in R$.
3. The addition is *associative*: $(a + b) + c = a + (b + c)$, for all $a, b, c \in R$.

4. There exists an *additive inverse* for each element: For each $a \in R$, there exists an element denoted $-a$ in R such that $a + (-a) = 0$.

5. The multiplication is *associative*: $a * (b * c) = (a * b) * c$, for all $a, b, c \in R$.

6. The operations distribute: $a * (b + c) = a * b + a * c$ and $(b + c) * a = b * a + c * a$, for all $a, b, c \in R$.

If a ring R has the additional property that the multiplication is commutative, then R is called a *commutative ring*. If R has the additional property of including a multiplicative identity, then R is called a *ring with unity*. The rings of interest to us are commutative rings with unity. In the event that a commutative ring R with unity includes multiplicative inverses for all of its nonzero elements, then R is a field.

Example 3.2.2. The integers $\mathbb{Z}$ under ordinary addition and multiplication form a commutative ring with unity, but you will recall that they do not form a field.✓

Example 3.2.3. The set $\mathbb{Z}_n$ of integers modulo n under addition and multiplication is a commutative ring with unity, but you will recall that $\mathbb{Z}_n$ is only a field in the case that n is a prime.✓

Example 3.2.4. The set $2\mathbb{Z}$ of even integers under ordinary addition and multiplication is a commutative ring without unity.✓

Example 3.2.5. The set $GF(2)[x]$ of all polynomials in the variable x with coefficients from the binary field $GF(2)$ under ordinary polynomial addition and multiplication is a commutative ring with unity polynomial $e(x) = 1$.✓

Example 3.2.6. The set $GF(2)[x]/(x^n - 1)$, which can be informally defined as the set of all polynomials of degree less than $n \geq 1$ in the variable x with coefficients from the binary field $GF(2)$ under polynomial addition and multiplication modulo $x^n - 1$, is a commutative ring with unity polynomial $e(x) = 1$.✓

Example 3.2.6 is the most important ring in coding theory, and we have already worked with the elements of this ring in the context of code polynomials. We will soon understand the mathematics behind the notation $GF(2)[x]/(x^n-1)$, as we delve deeper into the connections between cyclic codes and and the theory of rings and *ideals*.

Definition 3.2.7. 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 has *closure* since $a + b \in A$ whenever $a, b \in A$. We say that an ideal A *absorbs* elements from R, since $ra, ar \in A$ whenever $a \in A$ and $r \in R$.

Example 3.2.8. For any ring R, $\{0\}$ and R are ideals of R. This is trivial to check. ✓

Example 3.2.9. For any positive integer n, the set $n\mathbb{Z} = \{0, \pm n, \pm 2n \ldots\}$ is an ideal of the ring of integers $\mathbb{Z}$. To see this, first notice that for any $a, b \in n\mathbb{Z}$, we have $a = a'n$ and $b = b'n$ for some integers a' and b'. Then, $a - b = a'n - b'n = n(a' - b')$, which is clearly in $n\mathbb{Z}$. Next, notice that for any $r \in \mathbb{Z}$, we have $ar = a'nr = n(a'r)$, which is also clearly in $n\mathbb{Z}$. By the definition of ideal (and since $\mathbb{Z}$ is commutative), we have shown that $n\mathbb{Z}$ is an ideal in $\mathbb{Z}$.✓

Definition 3.2.10. 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* of a commutative ring R with unity if there exists $g \in A$ such that every element $a \in A$ can be written as rg for some $r \in R$.

Example 3.2.11. The ideal $n\mathbb{Z}$ of the integers from Example 3.2.9 is clearly a principle ideal with generator n.

Example 3.2.12. Let $\langle 3\rangle = \{3r | r \in \mathbb{Z}_{36}\}$. We will prove that $\langle 3\rangle$ is an ideal of the ring $\mathbb{Z}_{36}$. To do this, we must show that (1) $a - b \in \langle 3\rangle$ for all $a, b \in \langle 3\rangle$, and (2) $ar \in \langle 3\rangle$ whenever $a \in \langle 3\rangle$ and $r \in \mathbb{Z}_{36}$. (Since $\mathbb{Z}_{36}$ is commutative, we need not separately address $ra \in \mathbb{Z}_{36}$.) To prove (1), consider two arbitrary elements $a, b \in \langle 3\rangle$. These elements are of the form $a = 3a'$ and $b = 3b'$, for some $a', b' \in \mathbb{Z}_{36}$, by the definition of $\langle 3\rangle$. Then $a - b = 3a' - 3b' = 3(a' - b')$, where $a' - b'$ is some other element in $\mathbb{Z}_{36}$, since $a', b' \in \mathbb{Z}_{36}$ and therefore $-b' \in \mathbb{Z}_{36}$. So, again by the definition of $\langle 3\rangle$, $a - b \in \mathbb{Z}_{36}$. To prove (2), consider two arbitrary elements $a \in \langle 3\rangle$ and $r \in \mathbb{Z}_{36}$. Then $ar = (3a')r = 3(a'r)$, where $a'r \in \mathbb{Z}_{36}$ since $a', r \in \mathbb{Z}_{36}$. So, again by the definition of $\langle 3\rangle$, $ar \in \mathbb{Z}_{36}$ whenever $a \in \langle 3\rangle$ and $r \in \mathbb{Z}_{36}$. Therefore, $\langle 3\rangle$ is an ideal of the ring $\mathbb{Z}_{36}$.

Question 4. Show that $\langle x + 1\rangle$ is an ideal in $GF(2)[x]$, the ring of polynomials with binary coefficients. To do this, you must show that (1) $a(x) - b(x) \in \langle x + 1\rangle$ for all $a(x), b(x) \in \langle x + 1\rangle$, and (2) $a(x)r(x) \in \langle x + 1\rangle$ whenever $a(x) \in \langle x + 1\rangle$ and $r(x) \in GF(2)[x]$.

In fact, for any q-ary polynomial $p(x)$, $\langle p(x)\rangle$ is an ideal in $GF(q)[x]$.

Theorem 3.2.13. Let R be a ring and A be an ideal of R. Then the set of cosets $R/A = \{r + A | r \in R\}$ is a ring under the operations $(s + A) + (t + A) = s + t + A$ and $(s + A)(t + A) = st + A$. The operations $s + t$ and st are done according to the operation of the ring R.

Example 3.2.14. We have shown that $2\mathbb{Z}$ is an ideal of the ring $\mathbb{Z}$, so we can consider the ring of cosets $\mathbb{Z}/2\mathbb{Z} = \mathbb{Z}/\langle 2\rangle$. Elements of this ring are cosets of the form $r + 2\mathbb{Z}$ where $r \in \mathbb{Z}$. For example, elements include $0 + 2\mathbb{Z}$, $1 + 2\mathbb{Z}$, $23 + 2\mathbb{Z}$, and $-45 + 2\mathbb{Z}$. However, we will see that there are only two *distinct* cosets in the ring. First, consider the coset $23 + 2\mathbb{Z}$. This represents the set $\{23 + 2n | n \in \mathbb{Z}\}$. Since $23 = 1 + 2(11)$, we can write this set as $\{1 + 2(11) + 2n | n \in \mathbb{Z}\} = \{1 + 2(11 + n) | n \in \mathbb{Z}\}$ $= \{1 + 2m | m \in \mathbb{Z}\}$. This latter representation clearly represents the coset $1 + 2\mathbb{Z}$. In fact, any coset of the form $r + 2\mathbb{Z}$ where r is an odd integer is equivalent to the coset $1 + 2\mathbb{Z}$. Similarly, any coset of the form $r + 2\mathbb{Z}$ where r is an even integer is equivalent to the coset $0 + 2\mathbb{Z}$.

Now, consider adding two specific elements in this ring of cosets: $23 + 2\mathbb{Z}$ and $-45 + 2\mathbb{Z}$. Adding together these elements gives $-22 + 2\mathbb{Z}$, which represents the set $\{-22 + 2n | n \in \mathbb{Z}\} = \{0 + 2(-11 + n) | n \in \mathbb{Z}\} = \{0 + 2m | m \in \mathbb{Z}\}$, which is the coset $0 + 2\mathbb{Z}$. So, $(23 + 2\mathbb{Z}) + (-45 + 2\mathbb{Z}) = 0 + 2\mathbb{Z}$. Notice then that adding elements of the ring of cosets $\mathbb{Z}/2\mathbb{Z}$ is essentially the same thing as adding elements of $\mathbb{Z}$ modulo 2 (since $23 + (-45) \equiv 0 \pmod 2$).

The above discussion suggests the true fact that $\mathbb{Z}/2\mathbb{Z}$ is regarded as "isomorphic to," or essentially the same mathematical structure as, $\mathbb{Z}_2$.

Question 5. Compute $(24 + 2\mathbf{Z})(4 + 2\mathbf{Z})$ inside the ring of cosets $\mathbb{Z}/2\mathbf{Z}$.

Example 3.2.15. We showed above that $\langle 3\rangle = \{3r | r \in \mathbb{Z}_{36}\}$ is an ideal of the ring $\mathbb{Z}_{36}$. We can therefore consider the ring of cosets $\mathbb{Z}_{36}/\langle 3\rangle = \{r + \langle 3\rangle | r \in \mathbb{Z}_{36}\}$. The elements in this ring of cosets have the form $r + \langle 3\rangle$, where $r \in \mathbb{Z}_{36}$. For example, two specific elements are $12 + \langle 3\rangle$ and $28 + \langle 3\rangle$. Adding these elements together gives $4 + \langle 3\rangle$, since $12 + 28 \equiv 4 \pmod{36}$. However, let's look more closely at $4 + \langle 3\rangle$. This element represents the set $\{4 + 3n | n \in \mathbb{Z}_{36}\}$, which can be rewritten as $\{1 + 3 + 3n | n \in \mathbb{Z}_{36}\} = \{1 + 3m | m \in \mathbb{Z}_{36}\}$. This represents the coset $1 + \langle 3\rangle$. So, $(12 + \langle 3\rangle) + (28 + \langle 3\rangle) = 1 + \langle 3\rangle$. Notice then that adding elements in the ring $\mathbb{Z}_{36}/\langle 3\rangle$ is essentially the same thing as adding elements of $\mathbb{Z}_{36}$ under addition modulo 3. In fact, the only three distinct cosets in $\mathbb{Z}_{36}/\langle 3\rangle$ are $0 + \langle 3\rangle$, $1 + \langle 3\rangle$, and $2 + \langle 3\rangle$, and this ring is isomorphic to $\mathbb{Z}_3$.

Question 6. Compute $(10 + \langle 3\rangle)(5 + \langle 3\rangle)$ inside the ring of cosets $\mathbb{Z}_{36}/\langle 3\rangle$.

Example 3.2.16. Consider the ring of cosets $GF(2)[x]/(x^3 - 1)$. The elements of this ring have the form $r(x) + \langle x^3 - 1\rangle$, where $r(x) \in GF(2)[x]$. Two specific elements are $1 + x^2 + \langle x^3 - 1\rangle$ and $x + \langle x^3 - 1\rangle$. Multiplying these elements gives $(1 + x^2 + \langle x^3 - 1\rangle)(x + \langle x^3 - 1) = x + x^3 + \langle x^3 - 1\rangle$,

which represents the set $\{x + x^3 + (x^3 - 1)g(x) | g(x) \in GF(2)[x]\}$. We can rewrite this as the set $\{x + 1 + (x^3 - 1) + (x^3 - 1)g(x) | g(x) \in GF(2)[x]\} = \{x + 1 + (x^3 - 1)g'(x) | g'(x) \in GF(2)[x]\}$, which represents the coset $1 + x + \langle x^3 - 1\rangle$. Comparing this with Example 3.1.3, we see that multiplying two elements in the ring of cosets $GF(2)[x]/(x^3 - 1)$ is essentially equivalent to multiplying two polynomials modulo $x^3 - 1$.

Question 7. Use multiplication in the ring of cosets $GF(2)[x]/(x^3 - 1)$ to compute the product $(x + x^2 + \langle x^3 - 1\rangle)(x + \langle x^3 - 1\rangle)$, and then compare your result to the discussion in Example 3.1.3.

Example 3.2.17. We will now look at the more general ring of cosets defined by $GF(q)[x]/(x^n - 1) = \{f(x) + \langle x^n - 1\rangle | f(x) \in GF(q)[x]\}$. Elements of this ring of cosets have the form $f(x) + \langle x^n - 1\rangle$, where $f(x) \in GF(q)[x]$. Since $f(x) \in GF(q)[x]$, the Division Algorithm implies that we may write $f(x)$ in the form $q(x)(x^n - 1) + r(x)$, where $q(x)$ is the quotient and $r(x)$ is the remainder obtained when dividing $f(x)$ by $x^n - 1$. We can therefore write elements of $GF(q)[x]/(x^n - 1)$ as $q(x)(x^n - 1) + r(x) + \langle x^n - 1\rangle$, where $q(x), r(x) \in GF(q)[x]$. However, since $\langle x^n - 1\rangle$ is an ideal in $GF(q)[x]$, $q(x)(x^n - 1)$ is an element of $\langle x^n - 1\rangle$, by the absorption property of ideals. This allows a simplification to obtain $GF(q)[x]/(x^n - 1) = \{r(x) + \langle x^n - 1\rangle | r(x) \in GF(q)[x]\}$. By the Division Algorithm, $r(x)$ has degree less than n, and we can finally write $GF(q)[x]/(x^n - 1) = \{c_0 + c_1x + \ldots + c_{n-1}x^{n-1} + \langle x^n - 1\rangle | c_i \in GF(q)\}$. We can now understand that elements in the ring of cosets $GF(q)[x]/(x^n - 1)$ have the form $c_0 + c_1x + \ldots + c_{n-1}x^{n-1} + \langle x^n - 1\rangle$, where $c_i \in GF(q)$. The absorption property of ideals implies that adding and multiplying elements of $GF(q)[x]/(x^n - 1)$ is equivalent to adding and multiplying polynomials of degree less than n using modulo $x^n - 1$ polynomial arithmetic.

Example 3.2.17 is fundamental to the study of cyclic codes. We showed in Example 3.2.17 that technically, the elements of the ring of cosets $GF(q)[x]/(x^n - 1)$ have the form $r(x) + \langle x^n - 1\rangle$, where $r(x)$ is a polynomial of degree less than n. However, we will *abuse notation* and say that elements of $GF(q)[x]/(x^n - 1)$ are simply polynomials $r(x)$ of degree less than n and understand that we are working modulo $x^n - 1$.

3.3 Ideals and Cyclic Codes

The following theorem motivates our study of ideals:

Theorem 3.3.1. Cyclic codes of length n over $GF(q)$ correspond precisely to the ideals in the ring $GF(q)[x]/(x^n - 1)$.

Proof. Suppose C is a cyclic code of length n over $GF(q)$. Then, the corresponding set of code polynomials $I(C)$ is contained in $GF(q)[x]/(x^n - 1)$ since the polynomials are of degree less than n over $GF(q)$. By Definition 3.2.7, to show that $I(C)$ forms an ideal in $GF(q)[x]/(x^n - 1)$, we must show that (1) $c(x) - d(x) \in I(C)$ for any code polynomials $c(x)$ and $d(x)$ in $I(C)$; and (2) $r(x)c(x) \in I(C)$ for any polynomial $r(x) \in GF(q)[x]/(x^n - 1)$ and $c(x) \in I(C)$. By Theorem 3.1.4, every cyclic code C does indeed satisfy these two conditions.

On the other hand, suppose that I is an ideal in $GF(q)[x]/(x^n - 1)$. Then its elements are polynomials of degree less than n, and by Definition 3.2.7, its elements satisfy (1) $a(x) - b(x) \in I$ whenever $a(x), b(x) \in I$; and (2) $r(x)a(x) \in I$ whenever $r(x) \in GF(q)[x]/(x^n - 1)$ and $a(x) \in I$. Theorem 3.1.4 then shows that these conditions imply that the set of polynomials I represents the code polynomials of a cyclic code.

Therefore, cyclic codes of length n over $GF(q)$ are precisely the ideals in the ring $GF(q)[x]/(x^n - 1)$. □

Example 3.3.2. In Example 3.2.16, we considered the rings of cosets $GF(2)[x]/(x^3 - 1)$. A trivial example of an ideal in this ring is the entire ring itself. The elements in this ideal can be viewed

as all binary polynomials of degree less than 3: $0, 1, x, x^2, x+1, x^2+1, x^2+x, x^2+x+1, \}$. These polynomials correspond respectively to the binary codewords $\{000, 001, 010, 100, 011, 101, 110, 111\}$, which clearly form a trivial example of a binary cyclic code.

In order to study nontrivial cyclic codes, we will have to study nontrivial ideals of the ring $GF(q)[x]/(x^n - 1)$. The next theorem will help us find these:

Theorem 3.3.3. Let C be an $[n, k]$ cyclic code corresponding to an ideal $I(C)$ in $GF(q)[x]/(x^n - 1)$. The following statements are true:

1. There exists a unique *monic* polynomial $g(x) \in I(C)$ of minimal degree $r < n$. (A monic polynomial has a coefficient of 1 on its highest power term.) This polynomial is called the *generator polynomial* of C.
2. $I(C)$ is a principal ideal with generator $g(x)$, so that every code polynomial $c(x)$ 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 $GF(q)[x]$.
3. The generator polynomial $g(x)$ divides $x^n - 1$ in $GF(q)[x]$.

Proof. 1. Since there is at least one non-empty ideal $I(C)$ of $R_n = GF(q)[x]/(x^n - 1)$ (why?), and since the degree of polynomials is bounded below by 0, there is certainly at least one polynomial of minimum degree in $I(C)$. Since the scalar multiplication of any polynomial in an ideal $I(C)$ remains in $I(C)$, we can find a monic polynomial of minimum degree in $I(C)$; call it $g(x)$ of degree $r < n$.

Suppose that $h(x)$ is another monic polynomial of minimum degree d in $I(C)$. Since $I(C)$ is an ideal, $h(x) - g(x) \in I(C)$. Since these polynomials have the same degree and are monic, their difference must be of lower degree. However, this contradicts the minimality of r. Therefore, there cannot be two monic polynomials of minimum degree in $I(C)$.

2. We must show that every element in $I(C)$ can be written as a multiple of $g(x)$, where $g(x)$ is the unique monic polynomial of minimum degree in $I(C)$. Consider $f(x)$ an arbitrary element of $I(C)$, so that $f(x)$ is represented by a polynomial of degree less than n. By the Division Algorithm in $GF(q)[x]$ (and since $f(x)$ must have degree larger than the degree of $g(x)$), we can write $f(x) = q(x)g(x) + r(x)$, where $q(x), r(x) \in GF(q)[x]$ and $\deg r(x) < \deg g(x)$. Since $f(x)$ has degree less than n, clearly $q(x)$ and $r(x)$ (and $g(x)$) must also have degree less than n. Then, by Theorem 3.1.4, since $I(C)$ corresponds to a cyclic code and since $g(x)$ is a code polynomial, $q(x)g(x) \in I(C)$. Since $f(x)$ is also in $I(C)$, by the linearity of the code (or by the closure property of ideals), $f(x) - q(x)g(x) = r(x)$ is also in $I(C)$. However, $\deg r(x) < \deg g(x)$ contradicts the minimality of the degree of $g(x)$ in $I(C)$. Therefore, $r(x)$ must be equivalent to 0, implying that $f(x)$ is indeed a multiple of $g(x)$.

3. Suppose that the unique monic polynomial of minimum degree in $I(C)$ does not divide $x^n - 1$ in $GF(q)[x]$. By the Division Algorithm in $GF(q)[x]$, we can write $x^n - 1 = q(x)g(x) + r(x)$, where $q(x), r(x) \in GF(q)[x]$ and $\deg r(x) < \deg g(x)$. Since rearranging gives $r(x) = x^n - 1 - q(x)g(x)$ in $GF(q)[x]$, we see that in $GF(q)[x]/x^n - 1$, $r(x)$ is congruent to $-q(x)g(x)$. Since $g(x) \in I(C)$ and $-q(x) \in GF(q)[x]$, we have $-q(x)g(x) \in I(C)$, by the absorption property of ideals. Therefore, $r(x)$ must also be in $I(C)$. However, by the Division Algorithm, $\deg r(x) < \deg g(x)$, which contradicts the minimality of the degree of $g(x)$ in $I(C)$. Therefore, $r(x)$ must be 0, implying that $g(x)$ indeed divides $x^n - 1$.

□

Theorem 3.3.3 allows us to think of ideals and cyclic codes as equivalent and allows us to talk about the unique generator polynomial for a cyclic code. 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, not of minimal degree, or not divisors of the appropriate $x^n - 1$.

Example 3.3.4. Suppose we would like to generate a binary cyclic code C of length 4. According to Theorem 3.3.1, this code must correspond to an ideal $I(C)$ in the ring $GF(2)[x]/(x^4 - 1)$. Then, according to Theorem 3.3.3, the ideal $I(C)$ must be generated by a divisor of $x^4 - 1$ in $GF(2)[x]$. Using Gap or paper and pencil, we find that $g(x) = 1 + x$ is a divisor of $x^4 - 1$ in $GF(2)[x]$. Then, the ideal generated by $g(x)$ consists of the elements $\{0, 1+x, 1+x^2, x+x^2, 1+x^3, x+x^3, x^2+x^3, 1+x+x^2+x^3\}$. (Check this!). These elements correspond respectively to the binary code polynomials $\{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}$. Inspection shows that these code polynomials indeed form a binary cyclic code of length 4.

Question 8. Show that the ideal $1 + x^3$ in $GF(2)[x]/(x^4 - 1)$ generates the same binary cyclic code of length 4 as generated by $1 + x$ above. Explain why this does not contradict Theorem 3.3.3.

Since the ideals corresponding to cyclic codes are generated by divisors of $x^n - 1$, there are precisely as many q-ary cyclic codes of length n as there are divisors of $x^n - 1$ in $GF(q)[x]/(x^n - 1)$. More formally:

Theorem 3.3.5. There is a one-to-one correspondence between monic divisors of $x^n - 1$ of degree r in $GF(q)[x]$ and q-ary cyclic codes of length n and dimension $n - r$.

Proof. By Theorems 3.3.1 and 3.3.3, any q-ary $[n, k]$ cyclic code C is an ideal in $GF(q)[x]/(x^n - 1)$ 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 $x^n - 1$.

On the other hand, suppose $g(x)$ is a monic divisor of $x^n - 1$ in $GF(q)[x]$. Consider the code $C = \langle g(x) \rangle$. Clearly, the code polynomial $g(x)$ generates C. Suppose for contradiction that $h(x)$ is actually the generator polynomial for C, so that $h(x)$ is a monic polynomial of minimum degree r in C. Since $h(x) \in C = \langle g(x) \rangle$, $h(x)$ must be a polynomial multiple of $g(x)$ inside $GF(q)[x]/(x^n - 1)$, in other words, $h(x) = m(x)g(x)$ inside $GF(q)[x]/x^n - 1$. So, $h(x) = m(x)g(x) + s(x)(x^n - 1)$ inside $GF(q)[x]$. But, since $g(x)$ is a divisor of $x^n - 1$, we can write $g(x)t(x) = x^n - 1$, for some $t(x) \in GF(q)[x]$. So, $h(x) = g(x)m(x) + s(x)t(x)g(x)$. This implies $h(x) = g(x)(m(x) + s(x)t(x))$, or simplified, that $h(x) = g(x)a(x)$, where $a(x) \in GF(q)[x]$. This leaves two cases: Either the degree of $g(x)$ is less than the degree of $h(x)$, or these polynomials are scalar multiples of each other. In the first case, we arrive at a contradiction, since $h(x)$ was assumed to be of minimum degree in C. In the second case, since both polynomials were assumed to be monic, they must be equal. In either case, we can conclude that $g(x)$ is in fact the unique monic generator polynomial of minimum degree for C.

□

Theorem 3.3.5 facilitates listing all of the cyclic codes of a given length, n. If we can factor $x^n - 1$ over the appropriate alphabet, then the resulting factors (and combinations thereof) are equivalent to the generator polynomials for all cyclic codes of length n. To find an $[n, k]$ cyclic code, we need a factor of $x^n - 1$ of degree $n - k$.

Unfortunately, as n becomes larger than 3, factoring $x^n - 1$ over arbitrary alphabets (or even just over $GF(2)$), becomes increasingly more difficult. Fortunately, in this course, the factorizations of $x^n - 1$ will be provided or can be obtained using software.

Gap can factor $x^n - 1$ with over different fields. For example, we can factor $x^10 - 1$ over `GF(2)`.

```
gap> x := Indeterminate(GF(2), "x");;

gap> Display(Factors(x^7-1));
[ x+Z(2)^0, x^3+x+Z(2)^0, x^3+x^2+Z(2)^0 ]
```

Gap displays the factors as powers of $Z(n)$, which represents an element of order $n-1$ inside $\mathbb{Z}_n$. If n is prime, we can use the function `Int` to figure out what $Z(n)$ is equal to. For example,

```
gap> Int(Z(2));
1
```

shows that the above factors are $x+1$, x^3+x+1, and x^3+x^2+1.

Question 9. Use Gap to factor x^7-1 over $GF(2)$.

Question 10. Suppose that $g(x) = 1+x+x^2+x^4$ is the generator matrix for a length 7 binary cyclic code. What is the dimension of this code? Use Gap to confirm that $g(x)$ indeed divides x^7-1.

Example 3.3.6. In order to find all of the binary cyclic codes of length 3, we must factor x^3-1 into irreducible polynomials over $GF(2)$. In other words, we must factor x^3-1 into polynomials that can no longer be factored. It can be shown using Gap (or higher-level mathematics) that the irreducible factors of x^3-1 are $(x+1)$ and (x^2+x+1). Therefore, there are four possible generator polynomials: $g_0(x) = 1$, $g_1(x) = x+1$, $g_2(x) = x^2+x+1$, and $g_3(x) = x^3+1$. To obtain the length 3 code generated by the degree 1 polynomial $g_1(x) = x+1$, we must multiply $g_1(x)$ by every message polynomial corresponding to messages of length 2. We see $0 \cdot g_1(x) = 0$, $1 \cdot g_1(x) = x+1$, $x \cdot g_1(x) = x^2+x$, and $(x+1) \cdot g_1(x) = x^2+1$. Therefore, $g_1(x) = x+1$ generates the code $\{0, 1+x, x+x^2, 1+x^2\} = \{000, 110, 011, 101\}$. Similarly, $g_0(x) = 1$ generates the code which is equal to all of $GF(2)/(x^3-1)$ or all binary 3-tuples; $g_2(x) = x^2+x+1$ generates the code $\{0, 1+x+x^2\} = \{000, 111\}$; and $g_3(x) = x^3+1$ generates the code $\{0\} = \{000\}$. ✓

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

Example 3.3.7. Use Gap to find all possible generator polynomials for binary cyclic codes of length 5.

We can use Gap to generate cyclic codes given a generator polynomial. To declare a codeword polynomial, or in particular a generator polynomial, we must first declare the field in which the coefficients lie. For example, suppose we want a binary code of length 7 generated by generator polynomial $1+x+x^2$:

```
gap> x := Indeterminate(GF(2), "x");;

gap> gx := 1+x+x^2;;

gap> C := GeneratorPolCode(gx, 7, GF(2));

a cyclic [7,7,1]0 code defined by generator polynomial over GF(2)
```

Notice that since this code has minimum distance 1, it can correct 0 errors.

Question 11. Describe the set of codewords in the above code - how many codewords are there?

3.4 Generator and Parity Check Polynomials

In Example 3.3.6, we saw that the code $C = \{0, 1+x, x+x^2, 1+x^2\}$ is generated by $g_1(x) = x+1$. We can now expand upon this result and rewrite C 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; in other words, all codewords in C

are linear combinations of $g_1(x)$ and $xg_1(x)$. This implies that the corresponding vectors (110) and (011) can be used to form a generating matrix for C (recall Section 2.3). In particular, a generating matrix for C has the form:

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

where the first row is the vector corresponding to the generator polynomial $g_1(x)$, and the second row is the vector corresponding to its cyclic shift $xg_1(x)$. We next show that we can always use the generator polynomial to define the generator matrix of a cyclic code in this way.

Theorem 3.4.1. 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:

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

Proof. First, note that g_0 is nonzero: Otherwise, $(0, g_1, \ldots, g_{r-1}) \in C$ which implies that (g_1, …, 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 3.3.3 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 $GF(q)[x]$. 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 3.4.2. 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 $GF(2)[x]$. 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:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.\checkmark$$

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 3.3.3 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 12. 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) using the generator polynomial.

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 cannot 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 3.4.3. 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 13. Prove the above statement.

Given the generator matrix of a code, the function `CheckPol` returns the corresponding check polynomial. For example:

```
gap> H := CheckPol(GeneratorMatCode([[1,1,0], [0,1,1]], GF(2)));

x^2+x+Z(2)^0
```

You can also play around with Gap to confirm that given a code, `CheckPol` can also find the check polynomial.

Theorem 3.4.4. Suppose C is an $[n, k]$ cyclic code with parity check polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^k$. Then, a parity check matrix for C is the following $(n - k) \times n$ matrix:

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

The proof of this theorem uses the concept of dual codes, which are introduced in the special topics portion of this book.

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

$$\sum_{i=0}^{n-1} c_i h_{j-i} = 0, \text{ for } 0 \le j \le n-1. \tag{7}$$

The n equations represented in (7) 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 2.4.1 of a parity check matrix, we have shown that H is a parity check matrix for C. □

Example 3.4.5. The $[7, 3]$ code C constructed in Example 3.4.2 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:

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \checkmark$$

Question 14. What condition (equation) should the generator matrix from Example 3.4.2 and parity check matrix from Example 3.4.5 together satisfy? Check that these matrices do satisfy that equation.

We defined Hamming codes H_r in Section 2.7 by constructing their parity check matrices. We now show that these codes are equivalent to cyclic codes.

Theorem 3.4.6. The binary Hamming code H_r is equivalent to a cyclic code.

The proof of this theorem assumes knowledge of field theory and is provided for those students who have already taken a course in field theory.

Proof. Let $p(x)$ be an irreducible polynomial of degree r in $GF(2)[x]$. Then, the ring $GF(2)[x]/p(x)$ 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 $\alpha \in GF(2)[x]/p(x)$ such that $GF(2)[x]/p(x) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^r-2}\}$. Consider the matrix $H = [1, \alpha, \cdots, \alpha^{2^r-2}]$. Since each entry of H is an element of the field $GF(2)[x]/p(x)$ of order 2^r, we can express each entry α^i, $0 \leq i \leq 2^r - 1$ 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.*,

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

If $c(x) \in C$ and $r(x) \in GF(2)[x]/(x^n - 1)$, then $r(x)c(x) \in C$ because $r(\alpha)c(\alpha) = r(\alpha) \cdot 0 = 0$. It now follows from Theorem 3.1.4 that this Hamming code is cyclic. □

3.5 Answers to Chapter 3 Reading Questions

Answer 1. The corresponding code polynomial is $c(x) = 1 + x^2 + x^3$.

Answer 2. For a codeword of length 5, the maximum degree of the associated code polynomial is 4.

Answer 3. $(1 + x^2)(x^2) = x^2 + x^4 \equiv x^2 + x$ The codeword corresponding to this polynomial looks like 011, which is two right shifts of 101, the code polynomial we started with.

3.6 Chapter 3 Problems

Problem 3.1. Suppose you are working with an arbitrary cyclic code C. Let $m = (m_0, m_1, m_2)$ be a message that you want to encode using the generator polynomial $g(x) = g_0 + g_1x$. (Note that we

know g_0 is nonzero and g_1 is 1 since generator polynomials are monic, but do the problem out with the general notation.)

1. Encode the message using polynomial multiplication.
2. What are (n, k) for this code?
3. Use $g(x)$ to produce a non-systematic generator matrix G for C.
4. Now encode the same message m using the non-systematic generator matrix G.
5. Manipulate your answers to (a) and (d) to see that they are equal.

Problem 3.2. Let C be the binary cyclic code of length 9 with generator polynomial $g(x) = 1 + x^3$.

1. Find the parity check polynomial $h(x)$ for C.
2. Use $h(x)$ to confirm that 010100110 is a codeword in C.
3. Use $h(x)$ to find a (non-systematic) parity check matrix H for C.
4. Use H to again confirm that 010100110 is a codeword in C.
5. What message was encoded to 010100110?
6. What are (n, k) for the code C?

Problem 3.3. Find a basis for the smallest binary linear cyclic code of length 7 containing the codeword 1101000.

Problem 3.4. Show that the subring of rational numbers is not an ideal in the Reals.

Problem 3.5. 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.

Problem 3.6. Suppose that I is an ideal of R and that $1 \in I$. Prove that $I = R$

Problem 3.7. Show that every ideal in $\mathbb{Z}$ is a principal ideal.

Problem 3.8. Consider the polynomial ring $GF(2)[x]/(x^n - 1)$. Let $f(x), g(x)$ be two fixed polynomials in $GF(2)[x]/(x^n - 1)$. Consider the subset generated by these two fixed polynomials: $\langle f(x), g(x) \rangle = \{a(x)f(x) + b(x)g(x) \mid a(x), b(x) \in GF(2)[x]/(x^n - 1)\}$. Using the definition of an ideal, prove that this subset is an ideal of $GF(2)[x]/(x^n - 1)$.

Problem 3.9. Gap: List all of the distinct ideals in $GF(2)[x]/\langle (x^7 - 1) \rangle$ by listing their generators.

Problem 3.10. Gap: List by dimension all of the binary cyclic codes of length 31. Do the same for length 63 and 19.

4 Classes of Powerful Cyclic Codes

This Chapter uses more advanced abstract algebra to develop some of the most powerful error-control codes: BCH and Reed-Solomon (RS) 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 [5], and they were given their present name by W. Wesley Peterson in 1961 [4].

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*. In fact, RS codes are incorporated in the NASA Standard. These and several other applications of RS codes are described in [9].

4.1 Special Cases of BCH and RS Codes

In this section, we provide an introduction to the powerful classes of BCH and RS codes by focusing on restricted cases. The following sections develop the mathematical theory needed for the general theory of BCH and RS codes.

Definition 4.1.1. The multiplicative *order* of a non-zero field element α is defined as the smallest positive integer r such that $\alpha^r \equiv 1$.

Example 4.1.2. In the field $\mathbb{Z}_7$, the element 3 has order 6, since $3^6 \equiv 1$, but 3^x is not equivalent to 1 for any $x < 6$. (Confirm this by computing powers of 3 modulo 7)✓

Question 1. Find the order of 2 inside $\mathbb{Z}_7$.

We can also use Gap to find the order of a field element. For example, to find the multiplicative order of 3 inside $\mathbb{Z}_7$, we type:

```
gap> OrderMod(3, 7); 6
```

`OrderMod(<n>, <m>)` returns the multiplicative order of the integer n modulo m when n and m are relatively prime. Otherwise, the order is not defined and the command outputs 0.

Question 2. Use Gap to find the order of 2 inside $\mathbb{Z}_7$.

Theorem 4.1.3. (*The Modified BCH Bound*) In order to build a p-ary cyclic code C of length $p-1$, where p is prime, begin by finding an element α of order $p-1$ in $\mathbb{Z}_p$. Next, define a generator polynomial $g(x)$ for C as $(x-\alpha^b)(x-\alpha^{b+1})\cdots(x-\alpha^{b+\delta-2})$ for some integers $b \geq 0$ and $\delta \geq 1$. Hence, $g(x)$ contains $\delta-1$ consecutive powers of α as zeros or roots. Then, the cyclic code C defined by this generator polynomial $g(x)$ has minimum distance $d_{min} \geq \delta$.

The original BCH Bound, used to define general BCH codes, does not restrict the alphabet to be a prime field and does not restrict the length of the code to be one less than the size of the alphabet. Reed-Solomon (RS) codes are q^m-ary BCH codes of length $q^m - 1$. Our Modified BCH Bound will allow us to construct RS codes in the special case where the alphabet is a prime field.

Example 4.1.4. Let's find a generator polynomial for a 7-ary Reed-Solomon code of length 6 and minimum distance at least 4. First, we need to find an element $\alpha \in \mathbb{Z}_7$ with order 6. Example 4.1.2 shows that 3 satisfies our requirement. Since we need a minimum distance of 4, we need to build our generator polynomial so that it has three consecutive powers of 3 as its roots. We choose to define $g(x) = (x-3)(x-6)(x-2)$ (since $3^2 \equiv 6$ and $3^3 \equiv 2$). Multiplying out $g(x)$ over $\mathbb{Z}$ gives $g(x) = x^3 + 3x^2 + x + 6$. This polynomial is used to generate a $(6, 4, 4)$ Reed-Solomon code over $\mathbb{Z}_7$. Since the degree of $g(x)$ is 3 and the codelength is 6, the dimension is $6 - 3 = 3$. ✓

When using our Modified BCH Bound, we will always be in search of elements of order $p-1$ inside $\mathbb{Z}_p$, where p is prime. In mathematics, these elements are known as primitive roots, and the Gap command `PrimitiveRootMod(p)` finds these primitive roots for us. For example,

```
gap> PrimitiveRootMod(7);
3
```

Although there are often many primitive roots for a given prime, this Gap command will return the smallest such primitive root.

We can also use Gap to automate the construction of the generator matrix for a RS code. To illustrate, we will redo Example 4.1.4 using Gap:

```
gap> x := Indeterminate(GF(7), "x");;

gap> gx := (x-3^1)*(x-3^2)*(x-3^3);;

gap> C := GeneratorPolCode(gx, 6, GF(7));
a cyclic [6,3,1..4]2..3 code defined by generator polynomial over GF(7)
```

The first command defines the variable `x`. The second command defines the generator polynomial `gx` using three consecutive powers of 3. The third command tells Gap to generate a cyclic code of length 6 over $GF(7)$ using generator polynomial `gx`. The output confirms that we have generated a cyclic length 6 code of dimension 3. The minimum distance is between 1 and 4, but can determine the exact minimum distance via the following input:

```
gap> MinimumDistance(C);
4
```

Guava also has a built-in function for generating Reed-Solomon Codes. `ReedSolomonCode(<n>, <d>)` produces a length n RS code over $GF(n+1)$ with minimum distance d.

Question 3. Generate and compare the generator matrices for the above code `C` and the Gap-generated RS code of the same parameters.

It is quite simple to increase the minimum distance of a RS code. In order to increase the minimum distance of the RS code in Example 4.1.4 from 4 to 5, we need to include 2 more factors in the generator matrix.

Question 4. Write down a generator polynomial for a 7-ary RS code of length 6 and minimum distance 5.

Since the RS code in Example 4.1.4 has dimension $k = n - r = 6 - 3 = 3$,, this 7-ary code has $3^7 = 2187$ codewords.

Question 5. How many codewords does the RS code constructed in Question 4 have?

The preceding examples, comments, and questions highlight the trade-off that exists between minimum distance and total number of codewords. It should also be intuitively clear that by including fewer codewords, it is possible to increase the minimum distance between the codewords. Striking the right balance on this scale is of utmost importance in practical implementations.

Theorem 4.1.5. Reed-Solomon codes are maximum-distance separable (MDS); hence the parameters of an $[n, k, d]$ RS code satisfy the equation $d = n - k + 1$.

Proof. By the Singleton bound given in Theorem 2.6.2, 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 4.1.5 shows that RS codes satisfy the Singleton bound of Theorem 2.6.2 with equality. This means that given their length and dimension, RS codes are optimal in their distance properties. One notable property of MDS codes is that if a code is MDS, then so is its dual code, as defined in Section 5.1.

We can use the Gap `IsMDSCode` command to confirm that RS codes are MDS.

One of the advantages of BCH and RS codes is the ability to build codes with huge numbers of codewords. The (Modified) BCH Bound provides a very simple way to construct codes with optimal distance properties and many codewords. Importantly, there exist efficient decoding algorithms for these codes, making them very desirable in practice.

4.2 Minimal Polynomials

In order to study general BCH and RS codes, we need to learn how to find *minimal polynomials* of elements in a field. We begin with several definitions.

Definition 4.2.1. Let α be an element in $GF(q^m)$. The *minimal polynomial* of α with respect to the subfield $GF(q)$ is the monic polynomial $p(x) \in GF(q)[x]$ of minimal degree such that $p(\alpha) = 0$.

The above definition depends on the phrase "with respect to the subfield $GF(q)$," and it is meaningless without specifying a subfield.

The monic minimal polynomial of an element with respect to a subfield $GF(q)$ is unique: Suppose $f(x)$ and $g(x)$ are distinct monic minimal polynomials of $\alpha \in GF(q^m)$ with respect to $GF(q)$. 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 6. Prove that the minimal polynomial of an element is always irreducible.

Example 4.2.2. If $\beta \in GF(q)$, then its minimal polynomial with respect to $GF(q)$ 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 4.2.3. The minimal polynomial of $i \in \mathbb{C}$ with respect to the subfield $\mathbb{R}$ is $x^2 + 1$. ✓

Example 4.2.4. 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 $\alpha \in GF(q^m)$ and the subfield $GF(q)$. The analog of complex conjugation is the homomorphism $\phi : GF(q^m) \to GF(q^m)$ defined by $\phi : \alpha \mapsto \alpha^q$. This map fixes precisely the subfield $GF(q)$. We might guess that the minimal polynomial of an element $\alpha \in GF(q^m)$ with respect to $GF(q)$ 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 $\alpha \in GF(q^m)$ with respect to $GF(q)$ 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 4.2.5. Let α be an element in $GF(q^m)$. The *conjugates* of α with respect to the subfield $GF(q)$ are the elements α, α^q, $\alpha^{q^2}, \ldots$.

Again, the above definition depends on the phrase "with respect to the subfield $GF(q)$," and it is meaningless without specifying a subfield.

Example 4.2.6. If $\alpha \in GF(q)$, then its only conjugate with respect to $GF(q)$ is itself. ✓

Definition 4.2.7. The *conjugacy class* of $\alpha \in GF(q^m)$ with respect to the subfield $GF(q)$ is the set consisting of the distinct conjugates of α with respect to $GF(q)$. Note that α is always contained in its conjugacy class.

Theorem 4.2.8. The conjugacy class of $\alpha \in GF(q^m)$ with respect to the subfield $GF(q)$ contains v elements, where v divides m and v is the smallest integer such that $\alpha^{q^v} = \alpha$.

Proof. See [8]. □

Example 4.2.9. Let α be an element of order 3 in $GF(16)$. The conjugates of α with respect to $GF(2)$ are α, α^2, $\alpha^{2^2} = \alpha^{3+1} = \alpha$, etc. Hence, the conjugacy class with respect to $GF(2)$ of α is $\{\alpha, \alpha^2\}$.✓

Question 7. Convince yourself that the conjugacy class with respect to $GF(4)$ of α, an element of order 63 in $GF(64)$, is $\{\alpha, \alpha^4, \alpha^{16}\}$.

Theorem 4.2.10. Let α be an element in $GF(q^m)$. Let $p(x)$ be the minimal polynomial of α with respect to $GF(q)$. The roots of $p(x)$ are exactly the conjugates of α with respect to $GF(q)$.

Proof. See [8]. □

Example 4.2.11. It is possible to construct $GF(8)$ by identifying $GF(2)[x]/(x^3 + x + 1)$ with $GF(2)[\alpha]$, 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).

4.3 BCH and Reed-Solomon codes

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

Definition 4.3.1. Fix an integer n. Let $GF(q^m)$ be the smallest extension field of $GF(q)$ that contains an element of order n. Let β be an element of $GF(q^m)$ of order n. A cyclic code of length n over $GF(q)$ is called a *BCH code of designed distance* δ if its generator polynomial $g(x) \in GF(q)[x]$ 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 $GF(q^m)$, 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 4.3.2. An $n \times n$ *Vandermonde* matrix V is defined as follows:

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

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

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

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 4.3.3. *(BCH bound)* Let C be a BCH code of designed distance δ, with all notation as defined in Definition 4.3.1. 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

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

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

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

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

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

Since $c_{a_1}, \dots, c_{a_w}$ 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:

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

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 2.1.5, 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 $GF(q)$, follow the procedure described below.

1. Find an element β of order n in a field $GF(q^m)$, 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 $GF(q)$. (Each of the minimal polynomials should appear only once in the product).

Example 4.3.4. 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 $GF(32)$, 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 $GF(32)$ 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 redundancy 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.

One way to define RS codes is as a special case of nonbinary BCH codes:

Definition 4.3.5. 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 $GF(q)$. Hence, the conjugacy class of β with respect to $GF(q)$ consists of the unique element β. It follows that all $2t$ consecutive powers of β are also in $GF(q)$, and their conjugacy classes with respect to $GF(q)$ are also singletons. It follows from Theorem 4.2.10 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 $GF(q)$, 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 4.3.6. Let's construct a 2-error-correcting RS code over $GF(8)$. 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: $g(x) = (x-\beta)(x-\beta^2)(x-\beta^3)(x-\beta^4) = x^4 + \beta^3 x^3 + x^2 + \beta x + \beta^3$. Since $\deg g(x) = 4$, we have constructed a $[7,3]$ 2-error-correcting narrow-sense RS code over $GF(8)$. ✓

Theorem 4.3.7. ([8]) An $[n,k]$ Reed-Solomon code C has minimum distance $d = n-k+1$.

4.4 Chapter 4 Problems

Problem 4.1. Go through all of the necessary steps to build a generator polynomial for a 3-error correcting 11-ary Reed-Solomon code of length 10.

Problem 4.2. Let $n = 4$, $k = 2$, $F = \mathbb{Z}_5$, $\vec{v} = (1111)$, $\vec{\alpha} = (1234)$, and $C = GRS_{n,k}(\vec{\alpha}, \vec{v})$.

a) List the elements of C.

b) Find a generator matrix for C.

c) Find $d_{min}(C)$.

d) How many errors will C correct?

Problem 4.3. Recall that if $a \in \mathbb{Z}_p$ is nonzero, then there is a $b \in \mathbb{Z}_p$ such that $ab = 1$. We will denote b by a^{-1}. Let $F = \mathbb{Z}_p$ and $C = GRS_{n,k}(\vec{\alpha}, \vec{v})$ where, as usual, $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. Assume that $\alpha_i \neq 0$ for all $i = 1, 2, \ldots n$. Define $\vec{\beta} = (\alpha_1^{-1}, \alpha_2^{-1}, \ldots, \alpha_n^{-1})$. Find a vector $\vec{w}$ such that $C = GRS_{n,k}(\vec{\beta}, \vec{w})$.

Problem 4.4. 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 $\delta_2 \geq \delta_1$.

Problem 4.5. Write down a generator polynomial $g(x)$ for a 16-ary $[15, 11]$ Reed–Solomon code.

Problem 4.6. 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.

5 Special Topics

5.1 Dual Codes

The notion of dual codes is one of the most interesting topics in coding theory, however, it is also often confusing at first read. This section provides a brief introduction. You may want to re-read Section 2.4 before reading this section.

We will need the following definitions from linear algebra.

Definition 5.1.1. 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 1. 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 5.1.2. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal* if $\mathbf{u} \cdot \mathbf{v} = 0$.

Definition 5.1.3. 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 5.1.4. 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 5.1.5. 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 5.1.5 motivates the following definition:

Theorem 5.1.5 shows that C is equal to the nullspace of the parity check matrix H.

We have seen that there are several relationships among the generator matrix G of a linear code C, the parity check matrix H of a linear code C, and the dual code $C^\perp$. We recap:

1. H is the generator matrix of $C^\perp$, i.e. $C^\perp$ is the rowspace of H.
2. C is in the nullspace of H
3. The rows of G are orthogonal with the rows of H.

5.2 Group of a Code

Note: Much of this section was written by Ken Brown, Cornell University.

Recall that 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 5.2.1. 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 5.2.1, 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 5.2.2. 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 5.2.3. 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 [1]. □

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 5.2.2 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 2. 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 5.2.4. 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: $(0\,3\,1\,4)(2\,5)$.

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 3. 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 $\mathbf{x}\tau$ componentwise by

$$(\mathbf{x}\tau)_i = x_{\tau(i)} \tag{8}$$

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

$$\mathbf{x}\tau = (x_0, x_1, \ldots, x_{n-1})\tau = (x_{\tau(0)}, x_{\tau(1)}, \ldots, x_{\tau(n-1)}). \tag{9}$$

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

$$\mathbf{x}\tau = (x_0, x_1, x_2)\tau = (x_2, x_1, x_0), \tag{10}$$

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

Question 4. Prove the following "associative law:" For any vector $\mathbf{x}$ and any two permutations $\sigma, \tau \in \mathfrak{S}_n$,

$$(\mathbf{x}\sigma)\tau = \mathbf{x}(\sigma\tau), \tag{11}$$

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 (8). But you are likely to get confused, and even think that (11) is wrong, if you try to use (9).]

Define σ_n on $A_n = \{0, 1, \ldots, n-1\}$ as the permutation written with cycle notation as $\sigma_n = (01\ldots n-1)$. 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:

$$\begin{aligned} \mathbf{x}\sigma &= (x_0, x_1, \ldots, x_{n-1})\sigma && (12)\\ &= (x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(n-1)}) && (13)\\ &= (x_1, x_2, \ldots, x_{n-1}, x_0) && (14) \end{aligned}$$

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 5.2.5. 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 2.4.6 that every permutation of the n coordinates of the codewords of C sends C onto an equivalent $[n, k]$ code or onto itself.

Definition 5.2.6. 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 5. Prove that the group of a code of length n is a subgroup of $\mathfrak{S}_n$.

Example 5.2.7. 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 5.2.8. 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* [3, 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 $\mathbf{r} = \mathbf{x} + \mathbf{e}$ is received, and we know in principle that it is possible to decode $\mathbf{r}$ and recover the codeword $\mathbf{x}$. We also know that the syndrome $S(\mathbf{r}) = \mathbf{r}H^T$ 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 5.2.9. Suppose the syndrome $S(\mathbf{r}) = \mathbf{s}$ has weight $\leq t$. Then $\mathbf{e}' = \mathbf{0}$ (so the k information bits of $\mathbf{r}$ are correct) and $\mathbf{e}'' = \mathbf{s}$. We therefore have

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

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

$$\mathbf{s} = \mathbf{r}H^T = \mathbf{e}H^T = \mathbf{e}'A + \mathbf{e}''. \tag{15}$$

Consequently,

$$w(\mathbf{s}) \geq w(\mathbf{e}'A) - w(\mathbf{e}''). \tag{16}$$

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

$$w(\mathbf{e}'A) \geq 2t + 1 - w(\mathbf{e}'). \tag{17}$$

Combining inequalities (16) and (17), 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 (15), 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.

6 Solutions to Selected Problems

- **Sol.** 1.4 Here are the elements of C:

000	100	200	300	400
011	111	211	311	411
022	122	222	322	422
033	133	233	333	433
044	144	244	344	444

  The minimum distance of C is one.

- **Sol.** 1.6 Notice that to do this and correct a single error, the strings of length five must all differ in at least three places. Suppose we could write down eight strings of length five that all differed in at least three places. Then, they could have at most two places the same. Now, there must be at least four strings that have the same first digit (since you only have 0 and 1 to choose from and there are a total of 8 strings). Of those four, there must be at least two that have the same second digit (for the same reason as above). So, we have two strings that are the same in the first two places (let's just suppose for simplicity that they both start with 00). It follows that they must be different in the last three positions. Now, consider one of the other strings (remember, there were at least 4) that starts with 0. It must differ from the two strings starting with 00 in at least three places. So, of the last 4 positions, it must differ from each of the 00 strings in at least three positions. But since the 00 strings differ in the last three positions, this is impossible.

- **Sol.** 1.15 We first claim that either all the codewords in C begin with zero or exactly half of them begin with zero. Let $\vec{c_1}, \vec{c_2}, \ldots, \vec{c_k}$ be the rows of a generator matrix for C. If $k = 1$, then $C = \{\vec{0}, \vec{c_1}\}$, so our claim holds. If $k = 2$, then $C = \{\vec{0}, \vec{c_1}, \vec{c_2}, \vec{c_1} + \vec{c_2}\}$. Checking all the possibilities, you can see that either all elements of C begin with a zero or exactly two elements in C begin with a one, so the claim holds. Now we inductively show that our claim is true. Suppose the claim holds for $k = l$. Here's how you can show the claim is true for $k = l + 1$. If all the $\vec{c_i}$'s begin with a zero, then all elements in C begin with a zero, so the claim holds. On the other hand, suppose one of the $\vec{c_i}$'s begins with a one, say $\vec{c}_{l+1}$. Now,

  $C = \{\alpha_1\vec{c_1} + \alpha_2\vec{c_2} + \cdots + \alpha_l\vec{c_l} + \alpha_{l+1}\vec{c}_{l+1} | \alpha_i \in \mathbb{Z}_2\}$

  By assumption, the elements in the set

  $Z = \{\alpha_1\vec{c_1} + \alpha_2\vec{c_2} + \cdots + \alpha_l\vec{c_l} | \alpha_i \in \mathbb{Z}_2\}$

  either all begin with a zero or exactly half do. If all begin with a zero, then half of the elements in C begin with a zero. If exactly half of the elements of Z begin with a zero, then again we have half of the elements of C begin with zero, so the claim holds.

  Now, in the same way, we can show that in the ith position either all elements of C have a zero, or exactly half have a zero. The sum of the weights of all the elements in C is just the total number of ones in all the codewords. By our claim, in the ith position, there is at most $\frac{2^k}{2} = 2^{k-1}$ ones. (Note that C has 2^k elements.) Since there are n positions, the sum of the weights of all the elements in C is less than or equal to $n2^{k-1}$.

- **Sol.** 2.1

  1. $(101111), (000001), (110111) \in \mathbb{Z}_2^6$.

     Sol. *Suppose*

     $$c_1(101111) + c_2(000001) + c_3(110111) = (000000).$$

     Then, $c_1 + c_3 = 0$ *from the first coordinate.* $c_3 = 0$ *from the second coordinate.* $c_1 = 0$ *from the third coordinate.* $c_1 + c_3 = 0$ *from the fourth coordinate.* $c_1 + c_3 = 0$ *from the*

fifth coordinate. And $c_1 + c_2 + c_3 = 0$ *from the sixth coordinate. So,* $c_1 = c_2 = c_3 = 0$ *is the only solution to the above equation. It follows that* $(101111), (000001)$ *and* (110111) *are linearly independent.*

2. $(000), (101) \in \mathbb{Z}_2^3$.

   Sol. *Note that* $1(000)+0(101) = (000)$*, so* (000) *and* (101) *are not linearly independent.*

3. $(10000), (01001), (00111) \in \mathbb{Z}_2^5$.

   Sol. *Let*

   $$c_1(10000) + c_2(01001) + c_3(00111) = (00000).$$

   Then, $c_1 = 0$ *from the first coordinate.* $c_2 = 0$ *from the second coordinate. And* $c_3 = 0$ *from the third coordinate. So,* $c_1 = c_2 = c_3 = 0$ *is the only solution to the above equation. It follows that* $(10000), (01001)$ *and* (00111) *are linearly independent.*

4. $(0212), (0010), (2212) \in \mathbb{Z}_3^4$.

   Sol. *Let*

   $$c_1(0212) + c_2(0010) + c_3(2212) = (0000).$$

   Then, $2c_3 = 0$ *from the first coordinate* $2c_1 + 2c_3 = 0$ *from the second coordinate, and* $c_1+c_2+c_3 = 0$ *from the third coordinate. So,* $c_3 = 0$ *from the first equation. That implies that* $c_3 = 0$ *from the second equation. And, finally, that implies that* $c_2 = 0$ *from the third equation. It follows that the vectors are linearly independent.*

- **Sol.** 2.10 Let $(x_1, x_2, x_3, x_4, x_5, x_6), (y_1, y_2, y_3, y_4, y_5, y_6) \in \mathbb{Z}_2^6$. Then,
  $P((x_1, x_2, x_3, x_4, x_5, x_6) \oplus (y_1, y_2, y_3, y_4, y_5, y_6))$

  $$\begin{aligned} &= (x_3 \oplus y_3, x_4 \oplus y_4, x_1 \oplus y_1, x_5 \oplus y_5, x_2 \oplus y_2, x_6 \oplus y_6) \\ &= (x_3, x_4, x_1, x_5, x_2, x_6) \oplus (y_3, y_4, y_1, y_5, y_2, y_6) \\ &= P((x_1, x_2, x_3, x_4, x_5, x_6)) \oplus P((y_1, y_2, y_3, y_4, y_5, y_6)) \end{aligned}$$

  Also, if $c \in \mathbb{Z}_2^6$, then

  $$\begin{aligned} P(c(x_1, x_2, x_3, x_4, x_5, x_6)) &= P((cx_1, cx_2, cx_3, cx_4, cx_5, cx_6) \\ &= (cx_3, cx_4, cx_1, cx_5, cx_2, cx_6) \\ &= c(x_3, x_4, x_1, x_5, x_2, x_6) \\ &= cP((x_1, x_2, x_3, x_4, x_5, x_6)) \end{aligned}$$

  And it follows that P is linear.

- **Sol.** 2.11 We claim that C has p^k elements. Let $|C|$ denote the number of elements in C. Now, let $\vec{c_1}, \vec{c_2}, \ldots, \vec{c_k}$ be the rows for a generator matrix for C. Then

  $$C = \{\alpha_1\vec{c_1} + \cdots + \alpha_k\vec{c_k} | \alpha_i \in \mathbb{Z}_p\}$$

  It follows that $|C| \leq p^k$. Now, suppose $|C| < p^k$. Then we have that

  $$\alpha_1\vec{c_1} + \cdots + \alpha_k\vec{c_k} = \beta_1\vec{c_1} + \cdots + \beta_k\vec{c_k}$$

where the α's aren't all equal to the β's. But then

$$(\alpha_1 - \beta_1)\vec{c_1} + \cdots + (\alpha_k - \beta_k)\vec{c_k} = \vec{0}.$$

Since the $\vec{c_i}$'s are linearly independent, it follows that $\alpha_i = \beta_i$ for every i. But that is a contradiction. It follows that $|C| = p^k$.

- **Sol.** 2.16 Suppose $e, e' \in G$ such that $ae = ea = a$ and $ae' = e'a = a$ for every $a \in G$. Then, letting $a = e'$, we have $e = ee' = e'$.

- **Sol.** 2.17 Let $b, c \in G$ such that $ba = ab = e$ and $ca = ac = e$. Then, $b = be = b(ac) = (ba)c = ec = c$.

- **Sol.** 2.22

  1. Does there exist a code $C \subseteq A^8$ where C has exactly four codewords and such that C corrects 2 errors? If so, find such a C. If not, prove it.

     Sol. *Here is such a C:*

     $$C = \{00000000, 11111000, 00011111, 11100111\}$$

  2. Does there exist a code $C \subseteq A^8$ where C has exactly five codewords and such that C corrects 2 errors? If so, find such a C. If not, prove it.

     Sol. *This is not possible. I'll show this by contradiction. Suppose there were such a C. Then $C = \{\vec{a}, \vec{b}, \vec{c}, \vec{d}, \vec{e}\}$. Now, consider the first postion of each of the elements of C. There are only two choices, a zero or a one. Since C has five elements, there must be a subset of C of three elements whose elements all agree in the first position. Without loss of generality, assume that $\vec{a}, \vec{b}$, and $\vec{c}$ all agree in the first position. Since C corrects two errors, the minimum distance must be at least five. So, of the remaining seven positions, $\vec{a}$ and $\vec{b}$ can have at most two where they agree. Likewise, $\vec{a}$ and $\vec{c}$ have at most two positions where they agree, and $\vec{b}$ and $\vec{c}$ have at most two positions where they agree. But that is a total of at most six positions. Hence, there is at least one position where $\vec{a}$ and $\vec{b}$ are different, $\vec{a}$ and $\vec{c}$ are different and $\vec{b}$ and $\vec{c}$ are different. But, this is impossible. Hence, no such code exists.*

- **Sol.** 2.23

  1. Bob's first block of seven bits is 1110110. Alice sends him the syndrome 111. What does Bob's first block become after error correction?

     Sol. *Bob computes his syndrome:*

     $$\vec{s}_B = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

     Now he finds the difference between his syndrome and Alice's:

     $$\vec{s} = \vec{s}_B - \vec{s}_A = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The minimum weight vector that gives this syndrome is (0010000)*, so Bob takes this to be the error vector. He thus corrects his string to*

$$(1110110) - (0010000) = (1100110).$$

2. Bob's second block is 0000111. Alice sends him the syndrome 111. What does Bob's second block become after error correction?

   Sol. *Now Bob's syndrome is*

$$\vec{s}_B = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

   This is the same as the syndrome that Alice sends. The difference between these syndromes is the zero vector, so Bob concludes that there is no error in his string and he leaves it unchanged.

7 Bibliography

References

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

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

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

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

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

[6] C. E. Shannon, *A mathematical theory of communication*, Bell System Tech. J., 27 (1948), pp. 379–423, 623–656.

[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.