Notes on Coding Theory

J.I.Hall
Department of Mathematics
Michigan State University
East Lansing, MI 48824 USA

3 January 2003

Preface

These notes were written over a period of years as part of an advanced undergraduate/beginning graduate course on Algebraic Coding Theory at Michigan State University. They were originally intended for publication as a book, but that seems less likely now. The material here remains interesting, important, and useful; but, given the dramatic developments in coding theory during the last ten years, significant extension would be needed.

The oldest sections are in the Appendix and are over ten years old, while the newest are in the last two chapters and have been written within the last year. The long time frame means that terminology and notation may vary somewhat from one place to another in the notes. (For instance, $\mathbf{Z}_p$, $\mathbb{Z}_p$, and $\mathbb{F}_p$ all denote a field with p elements, for p a prime.)

There is also some material that would need to be added to any published version. This includes the graphs toward the end of Chapter 2, an index, and in-line references. You will find on the next page a list of the reference books that I have found most useful and helpful as well as a list of introductory books (of varying emphasis, difficulty, and quality).

These notes are not intended for broad distribution. If you want to use them in any way, please contact me.

Please feel free to contact me with any remarks, suggestions, or corrections:

`jhall@math.msu.edu`

For the near future, I will try to keep an up-to-date version on my web page:

`www.math.msu.edu\~jhall`

Jonathan I. Hall
3 August 2001

The notes were partially revised in 2002. A new chapter on weight enumeration was added, and parts of the algebra appendix were changed. Some typos were fixed, and other small corrections were made in the rest of the text. I particularly thank Susan Loepp and her Williams College students who went through the notes carefully and made many helpful suggestions.

I have been pleased and surprised at the interest in the notes from people who have found them on the web. In view of this, I may at some point reconsider publication. For now I am keeping to the above remarks that the notes are not intended for broad distribution.

Please still contact me if you wish to use the notes. And again feel free to contact me with remarks, suggestions, and corrections.

Jonathan I. Hall
3 January 2003

General References

R.E. Blahut, "Theory and practice of error control codes," Addison-Wesley, 1983. ISBN 0201101025

R.J. McEliece, "Theory of information and coding," 2nd edition, Cambridge University Press, 2002. ISBN 0521000955

J.H. van Lint, "Introduction to coding theory," 3rd edition, Graduate Texts in Mathematics **86**, Springer, 1999. ISBN 3540641335

V.S. Pless, W.C. Huffman, eds., and R.A. Brualdi, asst.ed., "Handbook of coding theory," volumes 1,2, Elsevier, 1998. ISBN 044450088X

F.J. MacWilliams and N.J.A. Sloane, "Theory of error-correcting codes," North-Holland, 1977. ISBN 0444851933

Introductory Books

D.R. Hankerson, D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger, and J.R. Wall, "Coding theory and cryptography: the essentials," second edition, Marcel Dekker, 2000. ISBN 0824704657

R. Hill, "A first course in coding theory," Oxford University Press, 1986. ISBN 0198538049

J.H. van Lint, "Coding theory," Lecture Notes in Mathematics **201**, Springer-Verlag, 1971. ISBN 3540054766

V. Pless, "Introduction to the theory of error-correcting codes," 3rd edition, Wiley, 1998. ISBN 0471190470

O. Pretzel, "Error-correcting codes and finite fields," Oxford University Press, 1992. ISBN 0198596782

S.A. Vanstone and P.C. van Oorschot, "An introduction to error correcting codes with applications," Kluwer Academic Publishers, 1989. ISBN 0792390172

Contents

Chapter 1

Introduction

Claude Shannon's 1948 paper "A Mathematical Theory of Communication" gave birth to the twin disciplines of information theory and coding theory. The basic goal is efficient and reliable communication in an uncooperative (and possibly hostile) environment. To be efficient, the transfer of information must not require a prohibitive amount of time and effort. To be reliable, the received data stream must resemble the transmitted stream to within narrow tolerances. These two desires will always be at odds, and our fundamental problem is to reconcile them as best we can.

At an early stage the mathematical study of such questions broke into the two broad areas. Information theory is the study of achievable bounds for communication and is largely probabilistic and analytic in nature. Coding theory then attempts to realize the promise of these bounds by models which are constructed through mainly algebraic means. Shannon was primarily interested in the information theory. Shannon's colleague Richard Hamming had been laboring on error-correction for early computers even before Shannon's 1948 paper, and he made some of the first breakthroughs of coding theory.

Although we shall discuss these areas as mathematical subjects, it must always be remembered that the primary motivation for such work comes from its practical engineering applications. Mathematical beauty can not be our sole gauge of worth. Throughout this manuscript we shall concentrate on the algebra of coding theory, but we keep in mind the fundamental bounds of information theory and the practical desires of engineering.

1.1 Basics of communication

Information passes from a source to a sink via a conduit or channel. In our view of communication we are allowed to choose exactly the way information is structured at the source and the way it is handled at the sink, but the behaviour of the channel is not in general under our control. The unreliable channel may take many forms. We may communicate through space, such as talking across

a noisy room, or through time, such as writing a book to be read many years later. The uncertainties of the channel, whatever it is, allow the possibility that the information will be damaged or distorted in passage. My conversation may be drowned out or my manuscript weather.

Of course in many situations you can ask me to repeat any information that you have not understood. This is possible if we are having a conversation (although not if you are reading my manuscript), but in any case this is not a particularly efficient use of time. ("What did you say?" "What?") Instead to guarantee that the original information can be recovered from a version that is not too badly corrupted, we add redundancy to our message at the source. Languages are sufficiently repetitive that we can recover from imperfect reception. When I lecture there may be noise in the hallway, or you might be unfamiliar with a word I use, or my accent could confuse you. Nevertheless you have a good chance of figuring out what I mean from the context. Indeed the language has so much natural redundancy that a large portion of a message can be lost without rendering the result unintelligible. When sitting in the subway, you are likely to see overhead and comprehend that "IF U CN RD THS U CN GT A JB."

Communication across space has taken various sophisticated forms in which coding has been used successfully. Indeed Shannon, Hamming, and many of the other originators of mathematical communication theory worked for Bell Telephone Laboratories. They were specifically interested in dealing with errors that occur as messages pass across long telephone lines and are corrupted by such things as lightening and crosstalk. The transmission and reception capabilities of many modems are increased by error handling capability embedded in their hardware. Deep space communication is subject to many outside problems like atmospheric conditions and sunspot activity. For years data from space missions has been coded for transmission, since the retransmission of data received faultily would be very inefficient use of valuable time. A recent interesting case of deep space coding occurred with the Galileo mission. The main antenna failed to work, so the possible data transmission rate dropped to only a fraction of what was planned. The scientists at JPL reprogrammed the onboard computer to do more code processing of the data before transmission, and so were able to recover some of the overall efficiency lost because of the hardware malfunction.

It is also important to protect communication across time from inaccuracies. Data stored in computer banks or on tapes is subject to the intrusion of gamma rays and magnetic interference. Personal computers are exposed to much battering, so often their hard disks are equipped with "cyclic redundancy checking" CRC to combat error. Computer companies like IBM have devoted much energy and money to the study and implementation of error correcting techniques for data storage on various mediums. Electronics firms too need correction techniques. When Phillips introduced compact disc technology, they wanted the information stored on the disc face to be immune to many types of damage. If you scratch a disc, it should still play without any audible change. (But you probably should not try this with your favorite disc; a really bad scratch can cause problems.) Recently the sound tracks of movies, prone to film

breakage and scratching, have been digitized and protected with error correction techniques.

There are many situations in which we encounter other related types of communication. Cryptography is certainly concerned with communication, however the emphasis is not on efficiency but instead upon security. Nevertheless modern cryptography shares certain attitudes and techniques with coding theory.

With source coding we are concerned with efficient communication but the environment is not assumed to be hostile; so reliability is not as much an issue. Source coding takes advantage of the statistical properties of the original data stream. This often takes the form of a dual process to that of coding for correction. In data compaction and compression[1] redundancy is removed in the interest of efficient use of the available message space. Data compaction is a form of source coding in which we reduce the size of the data set through use of a coding scheme that still allows the perfect reconstruction of the original data. Morse code is a well established example. The fact that the letter "e" is the most frequently used in the English language is reflected in its assignment to the shortest Morse code message, a single dot. Intelligent assignment of symbols to patterns of dots and dashes means that a message can be transmitted in a reasonably short time. (Imagine how much longer a typical message would be if "e" was represented instead by two dots.) Nevertheless, the original message can be recreated exactly from its Morse encoding.

A different philosophy is followed for the storage of large graphic images where, for instance, huge black areas of the picture should not be stored pixel by pixel. Since the eye can not see things perfectly, we do not demand here perfect reconstruction of the original graphic, just a good likeness. Thus here we use data compression, "lossy" data reduction as opposed to the "lossless" reduction of data compaction. The subway message above is also an example of data compression. Much of the redundancy of the original message has been removed, but it has been done in a way that still admits reconstruction with a high degree of certainty. (But not perfect certainty; the intended message might after all have been nautical in thrust: "IF YOU CANT RIDE THESE YOU CAN GET A JIB.")

Although cryptography and source coding are concerned with valid and important communication problems, they will only be considered tangentially in this manuscript.

One of the oldest forms of coding for error control is the adding of a parity check bit to an information string. Suppose we are transmitting strings composed of 26 bits, each a 0 or 1. To these 26 bits we add one further bit that is determined by the previous 26. If the initial string contains an even number of 1's, we append a 0. If the string has an odd number of 1's, we append a 1. The resulting string of 27 bits always contains an even number of 1's, that is, it has even parity. In adding this small amount of redundancy we have not compromised the information content of the message greatly. Of our 27 bits,

[1] We follow Blahut by using the two terms compaction and compression in order to distinguish lossless and lossy compression.

26 of them carry information. But we now have some error handling ability. If an error occurs in the channel, then the received string of 27 bits will have odd parity. Since we know that all transmitted strings have even parity, we can be sure that something has gone wrong and react accordingly, perhaps by asking for retransmission. Of course our error handling ability is limited to this possibility of detection. Without further information we are not able to guess the transmitted string with any degree of certainty, since a received odd parity string can result from a single error being introduced to any one of 27 different strings of even parity, each of which might have been the transmitted string. Furthermore there may have actually been more errors than one. What is worse, if two bit errors occur in the channel (or any even number of bit errors), then the received string will still have even parity. We may not even notice that a mistake has happened.

Can we add redundancy in a different way that allows us not only to detect the presence of bit errors but also to decide which bits are likely to be those in error? The answer is yes. If we have only two possible pieces of information, say 0 for "by sea" and 1 for "by land," that we wish to transmit, then we could repeat each of them three times — 000 or 111 . We might receive something like 101 . Since this is not one of the possible transmitted patterns, we can as before be sure that something has gone wrong; but now we can also make a good guess at what happened. The presence of two 1's but only one 0 points strongly to a transmitted string 111 plus one bit error (as opposed to 000 with two bit errors). Therefore we guess that the transmitted string was 111. This "majority vote" approach to decoding will result in a correct answer provided at most one bit error occurs.

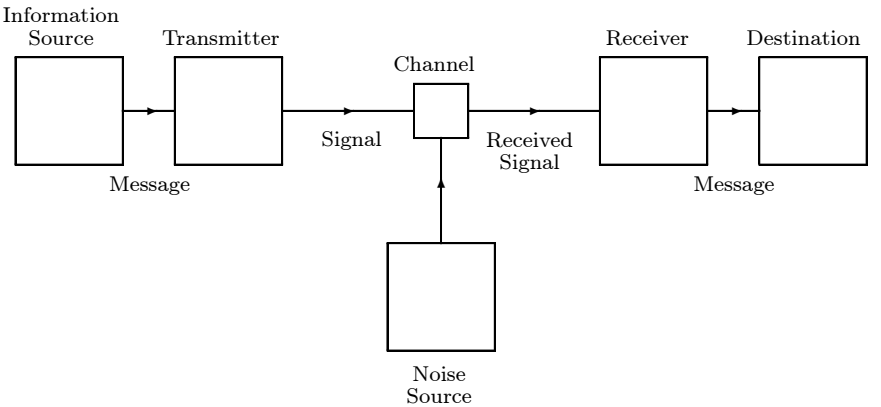
Now consider our channel that accepts 27 bit strings. To transmit each of our two messages, 0 and 1, we can now repeat the message 27 times. If we do this and then decode using "majority vote" we will decode correctly even if there are as many as 13 bit errors! This is certainly powerful error handling, but we pay a price in information content. Of our 27 bits, now only one of them carries real information. The rest are all redundancy.

We thus have two different codes of length 27 — the parity check code which is information rich but has little capability to recover from error and the repetition code which is information poor but can deal well even with serious errors. The wish for good information content will always be in conflict with the desire for good error performance. We need to balance the two. We hope for a coding scheme that communicates a decent amount of information but can also recover from errors effectively. We arrive at a first version of

> **The Fundamental Problem** — Find codes with both reasonable information content and reasonable error handling ability.

Is this even possible? The rather surprising answer is, "Yes!" The existence of such codes is a consequence of the Channel Coding Theorem from Shannon's 1948 paper (see Theorem 2.3.2 below). Finding these codes is another question. Once we know that good codes exist we pursue them, hoping to construct prac-

Figure 1.1: Shannon's model of communication



tical codes that solve more precise versions of the Fundamental Problem. This is the quest of coding theory.

1.2 General communication systems

We begin with Shannon's model of a general communication system, Figure 1.2. This setup is sufficiently general to handle many communication situations. Most other communication models, such as those requiring feedback, will start with this model as their base.

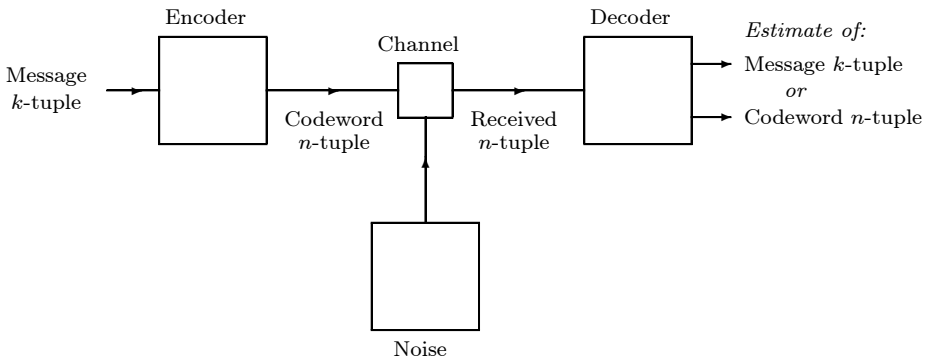
Our primary concern is block coding for error correction on a discrete memoryless channel. We next describe these and other basic assumptions that are made throughout this manuscript concerning various of the parts of Shannon's system; see Figure 1.2. As we note along the way, these assumptions are not the only ones that are valid or interesting; but in studying them we will run across most of the common issues of coding theory. We shall also honor these assumptions by breaking them periodically.

We shall usually speak of the transmission and reception of the words of the code, although these terms may not be appropriate for a specific envisioned application. For instance, if we are mainly interested in errors that affect computer memory, then we might better speak of storage and retrieval.

1.2.1 Message

Our basic assumption on messages is that each possible message k-tuple is as likely to be selected for broadcast as any other.

Figure 1.2: A more specific model



We are thus ignoring the concerns of source coding. Perhaps a better way to say this is that we assume source coding has already been done for us. The original message has been source coded into a set of k-tuples, each equally likely. This is not an unreasonable assumption, since lossless source coding is designed to do essentially this. Beginning with an alphabet in which different letters have different probabilities of occurrence, source coding produces more compact output in which frequencies have been levelled out. In a typical string of Morse code, there will be roughly the same number of dots and dashes. If the letter "e" was mapped to two dots instead of one, we would expect most strings to have a majority of dots. Those strings rich in dashes would be effectively ruled out, so there would be fewer legitimate strings of any particular reasonable length. A typical message would likely require a longer encoded string under this new Morse code than it would with the original. Shannon made these observations precise in his Source Coding Theorem which states that, beginning with an ergodic message source (such as the written English language), after proper source coding there is a set of source encoded k-tuples (for a suitably large k) which comprises essentially all k-tuples and such that different encoded k-tuples occur with essentially equal likelihood.

1.2.2 Encoder

block coding

We are concerned here with *block coding*. That is, we transmit blocks of symbols of fixed length n from a fixed alphabet A. These blocks are the codewords, and that codeword transmitted at any given moment depends only upon the present message, not upon any previous messages or codewords. Our encoder has no memory. We also assume that each codeword from the code (the set of all possible codewords) is as likely to be transmitted as any other.

Some work has been done on codes over mixed alphabets, that is, allowing the symbols at different coordinate positions to come from different alphabets. Such codes occur only in isolated situations, and we shall not be concerned with them at all.

Convolutional codes, trellis codes, lattice codes, and others come from encoders that have memory. We lump these together under the heading of *convolutional codes*. The message string arrives at the decoder continuously rather than segmented into unrelated blocks of length k, and the code string emerges continuously as well. That n-tuple of code sequence that emerges from the encoder while a given k-tuple of message is being introduced will depend upon previous message symbols as well as the present ones. The encoder "remembers" earlier parts of the message. The coding most often used in modems is of convolutional type.

convolutional codes

1.2.3 Channel

As already mentioned, we shall concentrate on coding on a *discrete memoryless channel* or DMC. The channel is discrete because we shall only consider finite alphabets. It is memoryless in that an error in one symbol does not affect the reliability of its neighboring symbols. The channel has no memory, just as above we assumed that the encoder has no memory. We can thus think of the channel as passing on the codeword symbol-by-symbol, and the characteristics of the channel can described at the level of the symbols.

discrete memoryless channel
DMC

An important example is furnished by the m-ary symmetric channel. The *m-ary symmetric channel* has input and output an alphabet of m symbols, say $x_1, \ldots, x_m$. The channel is characterized by a single parameter p, the probability that after transmission of symbol x_j the symbol $x_i \neq x_j$ is received. We write

m-ary symmetric channel

$$p(x_i|x_j) = p, \text{ for } i \neq j\,.$$

Related are the probability

$$s = (m-1)p$$

that after x_j is transmitted it is not received correctly and the probability

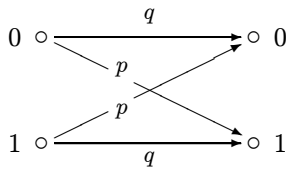
$$q = 1 - s = 1 - (m-1)p = p(x_j|x_j)$$

that after x_j is transmitted it is received correctly. We write $mSC(p)$ for the m-ary symmetric channel with *transition probability* p. The channel is symmetric in the sense $p(x_i|x_j)$ does not depend upon the actual values of i and j but only on whether or not they are equal. We are especially interested in the 2-ary symmetric channel or binary symmetric channel $BSC(p)$ (where $p = s$).

$mSC(p)$
transition probability
$BSC(p)$

Of course the signal that is actually broadcast will often be a measure of some frequency, phase, or amplitude, and so will be represented by a real (or complex) number. But usually only a finite set of signals is chosen for broadcasting, and the members of a finite symbol alphabet are modulated to the members of the finite signal set. Under our assumptions the modulator is thought of as part

Figure 1.3: The Binary Symmetric Channel



of the channel, and the encoder passes symbols of the alphabet directly to the channel.

Gaussian channel

There are other situations in which a continuous alphabet is the most appropriate. The most typical model is a *Gaussian channel* which has as alphabet an interval of real numbers (bounded due to power constraints) with errors introduced according to a Gaussian distribution.

The are also many situations in which the channel errors exhibit some kind of memory. The most common example of this is burst errors. If a particular symbol is in error, then the chances are good that its immediate neighbors are also wrong. In telephone transmission such errors occur because of lightening and crosstalk. A scratch on a compact disc produces burst errors since large blocks of bits are destroyed. Of course a burst error can be viewed as just one type of random error pattern and be handled by the techniques that we shall develop. We shall also see some methods that are particularly well suited to dealing with burst errors.

One final assumption regarding our channel is really more of a rule of thumb. We should assume that the channel machinery that carries out modulation, transmission, reception, and demodulation is capable of reproducing the transmitted signal with decent accuracy. We have a

Reasonable Assumption — Most errors that occur are not severe.

Otherwise the problem is more one of design than of coding. For a DMC we interpret the reasonable assumption as saying that an error pattern composed of a small number of symbol errors is more likely than one with a large number. For a continuous situation such as the Gaussian channel, this is not a good viewpoint since it is nearly impossible to reproduce a real number with perfect accuracy. All symbols are likely to be received incorrectly. Instead we can think of the assumption as saying that whatever is received should resemble to a large degree whatever was transmitted.

1.2.4 Received word

We assume that the decoder receives from the channel an n-tuple of symbols from the transmitter's alphabet A.

This assumption should perhaps be included in our discussion of the channel, since it really concerns the demodulator, which we think of as part of the channel

just as we do the modulator. We choose to isolate this assumption because it is a large factor in the split between block coding and convolutional coding. Many implementations in convolutional and related decoding instead combine the demodulator with the decoder in a single machine. This is the case with computer modems which serve as encoder/modulator and demodulator/decoder (MOdulator-DEModulator).

Think about how the demodulator works. Suppose we are using a binary alphabet which the modulator transmits as signals of amplitude $+1$ and -1. The demodulator receives signals whose amplitudes are then measured. These received amplitudes will likely not be exactly $+1$ or -1. Instead values like .750, and $-.434$ and .003 might be found. Under our assumptions each of these must be translated into a $+1$ or -1 before being passed on to the decoder. An obvious way of doing this is to take positive values to $+1$ and negative values to -1, so our example string becomes $+1, -1, +1$. But in doing so, we have clearly thrown away some information which might be of use to the decoder. Suppose in decoding it becomes clear that one of the three received symbols is certainly not the one originally transmitted. Our decoder has no way of deciding which one to mistrust. But if the demodulator's knowledge were available, the decoder would know that the last symbol is the least reliable of the three while the first is the most reliable. This improves our chances of correct decoding in the end.

In fact with our assumption we are asking the demodulator to do some initial, primitive decoding of its own. The requirement that the demodulator make precise (or hard) decisions about code symbols is called *hard quantization*. The alternative is *soft quantization*. Here the demodulator passes on information which suggests which alphabet symbol might have been received, but it need not make a final decision. At its softest, our demodulator would pass on the three real amplitudes and leave all symbol decisions to the decoder. This of course involves the least loss of information but may be hard to handle. A mild but still helpful form of soft quantization is to allow channel *erasures*. The channel receives symbols from the alphabet A but the demodulator is allowed to pass on to the decoder symbols from $A \cup \{?\}$, where the special symbol "?" indicates an inability to make an educated guess. In our three symbol example above, the decoder might be presented with the string $+1, -1, ?$, indicating that the last symbol was received unreliably. It is sometimes helpful to think of an erasure as a symbol error whose location is known.

hard quantization

soft quantization

erasures

1.2.5 Decoder

Suppose that in designing our decoding algorithms we know, for each n-tuple $\mathbf{y}$ and each codeword $\mathbf{x}$, the probability $p(\mathbf{y}|\mathbf{x})$ that $\mathbf{y}$ is received after the transmission of $\mathbf{x}$. The basis of our decoding is the following principle:

> **Maximum Likelihood Decoding** — When $\mathbf{y}$ is received, we must decode to a codeword $\mathbf{x}$ that maximizes $p(\mathbf{y}|\mathbf{x})$.

We often abbreviate this to **MLD**. While it is very sensible, it can cause problems similar to those encountered during demodulation. Maximum likelihood

MLD

decoding is "hard" decoding in that we must always decode to some codeword. This requirement is called *complete decoding*.

complete decoding

incomplete decoding

The alternative to complete decoding is *incomplete decoding*, in which we either decode a received n-tuple to a codeword or to a new symbol ∞ which could be read as "errors were detected but were not corrected" (sometimes abbreviated to "error detected"). Such *error detection* (as opposed to correction) can come about as a consequence of a *decoding default*. We choose this default alternative when we are otherwise unable (or unwilling) to make a sufficiently reliable decoding choice. For instance, if we were using a binary repetition code of length 26 (rather than 27 as before), then majority vote still deals effectively with 12 or fewer errors; but 13 errors produces a 13 to 13 tie. Rather than make an arbitrary choice, it might be better to announce that the received message is too unreliable for us to make a guess. There are many possible actions upon default. Retransmission could be requested. There may be other "nearby" data that allows an undetected error to be estimated in other ways. For instance, with compact discs the value of the uncorrected sound level can be guessed to be the average of nearby values. (A similar approach can be take for digital images.) We will often just declare "error detected but not corrected."

error detection

decoding default

Almost all the decoding algorithms that we discuss in detail will not be **MLD** but will satisfy **IMLD**, the weaker principle:

IMLD

> **Incomplete Maximum Likelihood Decoding** — When $\mathbf{y}$ is received, we must decode either to a codeword $\mathbf{x}$ that maximizes $p(\mathbf{y}|\mathbf{x})$ or to the "error detected" symbol ∞.

Of course, if we are only interested in maximizing our chance of successful decoding, then any guess is better than none; and we should use **MLD**. But this longshot guess may be hard to make, and if we are wrong then the consequences might be worse than accepting but recognizing failure. When correct decoding is not possible or advisable, this sort of error detection is much preferred over making an error in decoding. A *decoder error* has occurred if $\mathbf{x}$ has been transmitted, $\mathbf{y}$ received and decoded to a codeword $\mathbf{z} \neq \mathbf{x}$. A decoder error is much less desirable than a decoding default, since to the receiver it has the appearance of being correct. With detection we know something has gone wrong and can conceivably compensate, for instance, by requesting retransmission. Finally *decoder failure* occurs whenever we do not have correct decoding. Thus decoder failure is the combination of decoding default and decoder error.

decoder error

decoder failure

Consider a code C in A^n and a decoding algorithm $\mathbf{A}$. Then $\mathcal{P}_{\mathbf{x}}(\mathbf{A})$ is defined as the error probability (more properly, failure probability) that after $\mathbf{x} \in C$ is transmitted, it is received and not decoded correctly using $\mathbf{A}$. We then define

$$\mathcal{P}_C(\mathbf{A}) = |C|^{-1} \sum_{\mathbf{x} \in C} \mathcal{P}_{\mathbf{x}}(\mathbf{A}) \,,$$

the average error expectation for decoding C using the algorithm $\mathbf{A}$. This judges how good $\mathbf{A}$ is as an algorithm for decoding C. (Another good gauge would be the worst case expectation, $\max_{\mathbf{x} \in C} \mathcal{P}_{\mathbf{x}}(\mathbf{A})$.) We finally define the *error expectation* $\mathcal{P}_C$ for C via

error expectation $\mathcal{P}_C$

$$\mathcal{P}_C = \min_{\mathbf{A}} \mathcal{P}_C(\mathbf{A}) .$$

If $\mathcal{P}_C(\mathbf{A})$ is large then the algorithm is not good. If $\mathcal{P}_C$ is large, then no decoding algorithm is good for C; and so C itself is not a good code. In fact, it is not hard to see that $\mathcal{P}_C = \mathcal{P}_C(\mathbf{A})$, for every **MLD** algorithm $\mathbf{A}$. (It would be more consistent to call $\mathcal{P}_C$ the failure expectation, but we stick with the common terminology.)

We have already remarked upon the similarity of the processes of demodulation and decoding. Under this correspondence we can think of the detection symbol ∞ as the counterpart to the erasure symbol ? while decoder errors correspond to symbol errors. Indeed there are situations in concatenated coding where this correspondence is observed precisely. Codewords emerging from the "inner code" are viewed as symbols by the "outer code" with decoding error and default becoming symbol error and erasure as described.

A main reason for using incomplete rather than complete decoding is efficiency of implementation. An incomplete algorithm may be much easier to implement but only involve a small degradation in error performance from that for complete decoding. Again consider the length 26 repetition code. Not only are patterns of 13 errors extremely unlikely, but they require different handling than other types of errors. It is easier just to announce that an error has been detected at that point, and the the algorithmic error expectation $\mathcal{P}_C(\mathbf{A})$ only increases by a small amount.

1.3 Some examples of codes

1.3.1 Repetition codes

These codes exist for any length n and any alphabet A. A message consists of a letter of the alphabet, and it is encoded by being repeated n times. Decoding can be done by plurality vote, although it may be necessary to break ties arbitrarily.

The most fundamental case is that of binary repetition codes, those with alphabet $A = \{0, 1\}$. Majority vote decoding always produces a winner for binary repetition codes of odd length. The binary repetition codes of length 26 and 27 were discussed above.

1.3.2 Parity check and sum-0 codes

Parity check codes form the oldest family of codes that have been used in practice. The parity check code of length n is composed of all binary (alphabet $A = \{0, 1\}$) n-tuples that contain an even number of 1's. Any subset of $n - 1$ coordinate positions can be viewed as carrying the information, while the remaining position "checks the parity" of the information set. The occurrence of a single bit error can be detected since the parity of the received n-tuple will be odd rather than even. It is not possible to decide where the error occurred, but at least its presence is felt. (The parity check code is able to correct single erasures.)

The parity check code of length 27 was discussed above.

A versions of the parity check code can be defined in any situation where the alphabet admits addition. The code is then all n-tuples whose coordinate entries sum to 0. When the alphabet is the integers modulo 2, we get the usual parity check code.

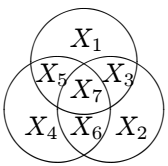
1.3.3 The $[7, 4]$ binary Hamming code

We quote from Shannon's paper:

> An efficient code, allowing complete correction of [single] errors and transmitting at the rate C [$=4/7$], is the following (found by a method due to R. Hamming):
>
> Let a block of seven symbols be $X_1, X_2, \ldots, X_7$ [each either 0 or 1]. Of these X_3, X_5, X_6, and X_7 are message symbols and chosen arbitrarily by the source. The other three are redundant and calculated as follows:
>
> $$\begin{array}{rcl} X_4 \text{ is chosen to make } \alpha & = & X_4 + X_5 + X_6 + X_7 \text{ even} \\ X_2 \text{ is chosen to make } \beta & = & X_2 + X_3 + X_6 + X_7 \text{ even} \\ X_1 \text{ is chosen to make } \gamma & = & X_1 + X_3 + X_5 + X_7 \text{ even} \end{array}$$
>
> When a block of seven is received, α, β, and γ are calculated and if even called zero, if odd called one. The binary number $\alpha\,\beta\,\gamma$ then gives the subscript of the X_i that is incorrect (if 0 then there was no error).

This describes a $[7, 4]$ binary Hamming code together with its decoding. We shall give the general versions of this code and decoding in a later chapter.

R.J. McEliece has pointed out that the $[7, 4]$ Hamming code can be nicely thought of in terms of the usual Venn diagram:



The message symbols occupy the center of the diagram, and each circle is completed to guarantee that it contains an even number of 1's (has even parity). If, say, received circles A and B have odd parity but circle C has even parity, then the symbol within $A \cap B \cap \overline{C}$ is judged to be in error at decoding.

1.3.4 An extended binary Hamming code

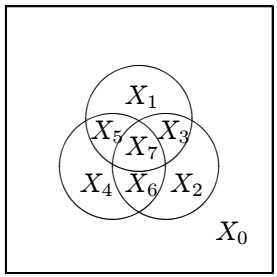
An extension of a binary Hamming code results from adding at the beginning of each codeword a new symbol that checks the parity of the codeword. To the

$[7,4]$ Hamming code we add an initial symbol:

$$X_0 \text{ is chosen to make } X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \text{ even}$$

The resulting code is the $[8,4]$ extended Hamming code. In the Venn diagram the symbol X_0 checks the parity of the universe.

The extended Hamming code not only allows the correction of single errors (as before) but also detects double errors.



1.3.5 The $[4,2]$ ternary Hamming code

This is a code of nine 4-tuples $(a,b,c,d) \in A^4$ with ternary alphabet $A = \{0,1,2\}$. Endow the set A with the additive structure of the integers modulo 3. The first two coordinate positions a, b carry the 2-tuples of information, each pair $(a,b) \in A^2$ exactly once (hence nine codewords). The entry in the third position is sum of the previous two (calculated, as we said, modulo 3):

$$a + b = c \ ,$$

for instance, with $(a,b) = (1,0)$ we get $c = 1 + 0 = 1$. The final entry is then selected to satisfy

$$b + c + d = 0 \ ,$$

so that $0 + 1 + 2 = 0$ completes the codeword $(a,b,c,d) = (1,0,1,2)$. These two equations can be interpreted as making ternary parity statements about the codewords; and, as with the binary Hamming code, they can then be exploited for decoding purposes. The complete list of codewords is:

$$\begin{array}{lll} (0,0,0,0) & (1,0,1,2) & (2,0,2,1) \\ (0,1,1,1) & (1,1,2,0) & (2,1,0,2) \\ (0,2,2,2) & (1,2,0,1) & (2,2,1,0) \end{array}$$

(1.3.1) PROBLEM. *Use the two defining equations for this ternary Hamming code to describe a decoding algorithm that will correct all single errors.*

1.3.6 A generalized Reed-Solomon code

We now describe a code of length $n = 27$ with alphabet the field of real number $\mathbb{R}$. Given our general assumptions this is actually a nonexample, since the alphabet is not discrete or even bounded. (There are, in fact, situations where these generalized Reed-Solomon codes with real coordinates have been used.)

Choose 27 distinct real numbers $\alpha_1, \alpha_2, \ldots, \alpha_{27}$. Our message k-tuples will be 7-tuples of real numbers $(f_0, f_1, \ldots, f_6)$, so $k = 7$. We will encode a given message 7-tuple to the codeword 27-tuple

$$\mathbf{f} = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_{27}))\,,$$

where

$$f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + f_4 x^4 + f_5 x^5 + f_6 x^6$$

is the polynomial function whose coefficients are given by the message. Our Reasonable Assumption says that a received 27-tuple will resemble the codeword transmitted to a large extent. If a received word closely resembles each of two codewords, then they also resemble each other. Therefore to achieve a high probability of correct decoding we would wish pairs of codewords to be highly dissimilar.

The codewords coming from two different messages will be different in those coordinate positions i at which their polynomials $f(x)$ and $g(x)$ have different values at α_i. They will be equal at coordinate position i if and only if α_i is a root of the difference $h(x) = f(x) - g(x)$. But this can happen for at most 6 values of i since $h(x)$ is a nonzero polynomial of degree at most 6. Therefore:

> distinct codewords differ in at least 21 $(= 27 - 6)$ coordinate positions.

Thus two distinct codewords are highly different. Indeed as many up to 10 errors can be introduced to the codeword $\mathbf{f}$ for $f(x)$ and the resulting word will still resemble the transmitted codeword $\mathbf{f}$ more than it will any other codeword.

The problem with this example is that, given our inability in practice to describe a real number with arbitrary accuracy, when broadcasting with this code we must expect almost all symbols to be received with some small error — 27 errors every time! One of our later objectives will be to translate the spirit of this example into a more practical setting.

Chapter 2

Sphere Packing and Shannon's Theorem

In the first section we discuss the basics of block coding on the m-ary symmetric channel. In the second section we see how the geometry of the codespace can be used to make coding judgements. This leads to the third section where we present some information theory and Shannon's basic Channel Coding Theorem.

2.1 Basics of block coding on the mSC

Let A be any finite set. A *block code* or *code*, for short, will be any nonempty subset of the set A^n of n-tuples of elements from A. The number $n = n(C)$ is the *length* of the code, and the set A^n is the *codespace*. The number of members in C is the *size* and is denoted $|C|$. If C has length n and size $|C|$, we say that C is an $(n, |C|)$ *code*.

block code

length
codespace
size
$(n, |C|)$ code

The members of the codespace will be referred to as *words*, those belonging to C being *codewords*. The set A is then the *alphabet*.

words
codewords
alphabet

If the alphabet A has m elements, then C is said to be an *m-ary code*. In the special case $|A|$=2 we say C is a *binary* code and usually take $A = \{0, 1\}$ or $A = \{-1, +1\}$. When $|A|$=3 we say C is a *ternary* code and usually take $A = \{0, 1, 2\}$ or $A = \{-1, 0, +1\}$. Examples of both binary and ternary codes appeared in Section 1.3.

m-ary code
binary
ternary

For a discrete memoryless channel, the Reasonable Assumption says that a pattern of errors that involves a small number of symbol errors should be more likely than any particular pattern that involves a large number of symbol errors. As mentioned, the assumption is really a statement about design.

On an $mSC(p)$ the probability $p(\mathbf{y}|\mathbf{x})$ that $\mathbf{x}$ is transmitted and $\mathbf{y}$ is received is equal to $p^d q^{n-d}$, where d is the number of places in which $\mathbf{x}$ and $\mathbf{y}$ differ. Therefore

$$p(\mathbf{y}|\mathbf{x}) = q^n (p/q)^d \,,$$

a decreasing function of d provided $q > p$. Therefore the Reasonable Assumption is realized by the $mSC(p)$ subject to

$$q = 1 - (m-1)p > p$$

or, equivalently,

$$1/m > p\ .$$

We interpret this restriction as the sensible design criterion that after a symbol is transmitted it should be more likely for it to be received as the correct symbol than to be received as any particular incorrect symbol.

EXAMPLES.

(i) Assume we are transmitting using the the binary Hamming code of Section 1.3.3 on $BSC(.01)$. Comparing the received word 0011111 with the two codewords 0001111 and 1011010 we see that

$$p(0011111|0001111) = q^6p^1 \approx .009414801\,,$$

while

$$p(0011111|1011010) = q^4p^3 \approx .000000961\,;$$

therefore we prefer to decode 0011111 to 0001111. Even this event is highly unlikely, compared to

$$p(0001111|0001111) = q^7 \approx .932065348\,.$$

(ii) If $m = 5$ with $A = \{0,1,2,3,4\}^6$ and $p = .05 < 1/5 = .2$, then $q = 1 - 4(.05) = .8$; and we have

$$p(011234|011234) = q^6 = .262144$$

and

$$p(011222|011234) = q^4p^2 = .001024\,.$$

For $\mathbf{x}, \mathbf{y} \in A^n$, we define

$$d_H(\mathbf{x},\mathbf{y}) = \text{the number of places in which } \mathbf{x} \text{ and } \mathbf{y} \text{ differ.}$$

Hamming distance

This number is the *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$. The Hamming distance is a genuine metric on the codespace A^n. It is clear that it is symmetric and that $d_H(\mathbf{x},\mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$. The Hamming distance $d_H(\mathbf{x},\mathbf{y})$ should be thought of as the number of errors required to change $\mathbf{x}$ into $\mathbf{y}$ (or, equally well, to change $\mathbf{y}$ into $\mathbf{x}$).

EXAMPLE.

$$d_H(0011111, 0001111) = 1\,;$$
$$d_H(0011111, 1011010) = 3\,;$$
$$d_H(011234, 011222) = 2\,.$$

(2.1.1) PROBLEM. *Prove the triangle inequality for the Hamming distance:*

$$d_H(\mathbf{x},\mathbf{y}) + d_H(\mathbf{y},\mathbf{z}) \geq d_H(\mathbf{x},\mathbf{z})\,.$$

The arguments above show that, for an $mSC(p)$ with $p < 1/m$, maximum likelihood decoding becomes:

> **Minimum Distance Decoding** — When $\mathbf{y}$ is received, we must decode to a codeword $\mathbf{x}$ that minimizes the Hamming distance $d_H(\mathbf{x}, \mathbf{y})$.

We abbreviate *minimum distance decoding* as **MDD**. In this context, incomplete decoding is incomplete minimum distance decoding **IMDD**:

minimum distance decoding
MDD
IMDD

> **Incomplete Minimum Distance Decoding** — When $\mathbf{y}$ is received, we must decode either to a codeword $\mathbf{x}$ that minimizes the Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ or to the "error detected" symbol ∞.

(2.1.2) PROBLEM. *Prove that, for an $mSC(p)$ with $p = 1/m$, every complete algorithm is an* **MLD** *algorithm.*

(2.1.3) PROBLEM. *Give a definition of what might be called* maximum distance decoding*,* **MxDD***; and prove that* **MxDD** *algorithms are* **MLD** *algorithms for an $mSC(p)$ with $p > 1/m$.*

In A^n, the *sphere*[1] of radius ρ centered at $\mathbf{x}$ is

sphere

$$S_\rho(\mathbf{x}) = \{\, \mathbf{y} \in A^n \mid d_H(\mathbf{x}, \mathbf{y}) \leq \rho \,\}.$$

Thus the sphere of radius ρ around $\mathbf{x}$ is composed of those $\mathbf{y}$ that might be received if at most ρ symbol errors were introduced to the transmitted codeword $\mathbf{x}$.

The volume of a sphere of radius ρ is independent of the location of its center.

(2.1.4) PROBLEM. *Prove that in A^n with $|A| = m$, a sphere of radius e contains $\sum_{i=0}^{e} \binom{n}{i} (m-1)^i$ words.*

For example, a sphere of radius 2 in $\{0,1\}^{90}$ has volume

$$1 + \binom{90}{1} + \binom{90}{2} = 1 + 90 + 4005 = 4096 = 2^{12}$$

corresponding to a center, 90 possible locations for a single error, and $\binom{90}{2}$ possibilities for a double error. A sphere of radius 2 in $\{0,1,2\}^8$ has volume

$$1 + \binom{8}{1}(3-1)^1 + \binom{8}{2}(3-1)^2 = 1 + 16 + 112 = 129\ .$$

For each nonnegative real number ρ we define a decoding algorithm $\mathbf{SS}_\rho$ for A^n called *sphere shrinking.*

$\mathbf{SS}_\rho$
sphere shrinking

[1] Mathematicians would prefer to use the term 'ball' here in place of ' sphere', but we stick with the traditional coding terminology.

Radius ρ Sphere Shrinking — If $\mathbf{y}$ is received, we decode to the codeword $\mathbf{x}$ if $\mathbf{x}$ is the unique codeword in $S_\rho(\mathbf{y})$, otherwise we declare a decoding default..

Thus $\mathbf{SS}_\rho$ shrinks the sphere of radius ρ around each codeword to its center, throwing out words that lie in more than one such sphere.

The various distance determined algorithms are completely described in terms of the geometry of the codespace and the code rather than by the specific channel characteristics. In particular they no longer depend upon the transition parameter p of an $mSC(p)$ being used. For **IMDD** algorithms **A** and **B**, if $\mathcal{P}_C(\mathbf{A}) \leq \mathcal{P}_C(\mathbf{B})$ for some $mSC(p)$ with $p < 1/m$, then $\mathcal{P}_C(\mathbf{A}) \leq \mathcal{P}_C(\mathbf{B})$ will be true for all $mSC(p)$ with $p < 1/m$. The **IMDD** algorithms are (incomplete) maximum likelihood algorithms on every $mSC(p)$ with $p \leq 1/m$, but this observation now becomes largely motivational.

EXAMPLE. Consider the specific case of a binary repetition code of length 26. Notice that since the first two possibilities are not algorithms but classes of algorithms there are choices available.

w = number of 1's	0	$1 \leq w \leq 11$	$= 12$	$= 13$	$= 14$	$15 \leq w \leq 25$	26
IMDD	$\mathbf{0}/\infty$	$\mathbf{0}/\infty$	$\mathbf{1}/\infty$	$\mathbf{0/1}/\infty$	$\mathbf{1}/\infty$	$\mathbf{1}/\infty$	$\mathbf{1}/\infty$
MDD	**0**	**0**	**0**	**0/1**	**1**	**1**	**1**
$\mathbf{SS}_{12}$	**0**	**0**	**0**	∞	**1**	**1**	**1**
$\mathbf{SS}_{11}$	**0**	**0**	∞	∞	∞	**1**	**1**
$\mathbf{SS}_0$	**0**	∞	∞	∞	∞	∞	**1**

Here **0** and **1** denote, respectively, the 26-tuple of all 0's and all 1's. In the fourth case, we have less error correcting power. On the other hand we are less likely to have a decoder error, since 15 or more symbol errors must occur before a decoder error results. The final case corrects no errors, but detects nontrivial errors except in the extreme case where all symbols are received incorrectly, thereby turning the transmitted codeword into the other codeword.

The algorithm $\mathbf{SS}_0$ used in the example is the usual error detection algorithm: when $\mathbf{y}$ is received, decode to $\mathbf{y}$ if it is a codeword and otherwise decode to ∞, declaring that an error has been detected.

2.2 Sphere packing

minimum distance

The code C in A^n has *minimum distance* $d_{min}(C) = d(C)$ equal to the minimum of $d_H(\mathbf{x}, \mathbf{y})$, as $\mathbf{x}$ and $\mathbf{y}$ vary over all distinct pairs of codewords from C. (This leaves some confusion over $d(C)$ for a length n code C with only one word. It may be convenient to think of it as any number larger than n.) An (n, M)-code with minimum distance d will sometimes be referred to as an (n, M, d)-*code*.

(n, M, d)-code

EXAMPLE. The minimum distance of the repetition code of length n is clearly n. For the parity check code any single error produces a word of

odd parity, so the minimum distance is 2. The length 27 generalized Reed-Solomon code of Example 1.3.6 was shown to have minimum distance 21.

Laborious checking reveals that the $[7, 4]$ Hamming code has minimum distance 3, and its extension has minimum distance 4. The $[4, 2]$ ternary Hamming code also has minimum distance 3. We shall see later how to find the minimum distance of these codes easily.

(2.2.1) LEMMA. *The following are equivalent for the code C in A^n:*

(1) *under $\mathbf{SS}_e$ any occurrence of e or fewer symbol errors will always be successfully corrected;*

(2) *for all distinct $\mathbf{x}, \mathbf{y}$ in C, we have $S_e(\mathbf{x}) \cap S_e(\mathbf{y}) = \emptyset$;*

(3) *the minimum distance of C, $d_{min}(C)$, is at least $2e + 1$.*

PROOF. Assume (1), and let $\mathbf{z} \in S_e(\mathbf{x})$, for some $\mathbf{x} \in C$. Then by assumption $\mathbf{z}$ is decoded to $\mathbf{x}$ by $\mathbf{SS}_e$. Therefore there is no $\mathbf{y} \in C$ with $\mathbf{y} \neq \mathbf{x}$ and $\mathbf{z} \in S_e(\mathbf{y})$, giving (2).

Assume (2), and let $\mathbf{z}$ be a word that results from the introduction of at most e errors to the codeword $\mathbf{x}$. By assumption $\mathbf{z}$ is not in $S_e(\mathbf{y})$ for any $\mathbf{y}$ of C other than $\mathbf{x}$. Therefore, $S_e(\mathbf{z})$ contains $\mathbf{x}$ and no other codewords; so $\mathbf{z}$ is decoded to $\mathbf{x}$ by $\mathbf{SS}_e$, giving (1).

If $\mathbf{z} \in S_e(\mathbf{x}) \cap S_e(\mathbf{y})$, then by the triangle inequality we have $d_H(\mathbf{x}, \mathbf{y}) \leq d_H(\mathbf{x}, \mathbf{z}) + d_H(\mathbf{z}, \mathbf{y}) \leq 2e$, so (3) implies (2).

It remains to prove that (2) implies (3). Assume $d_{min}(C) = d \leq 2e$. Choose $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ in C with $d_H(\mathbf{x}, \mathbf{y}) = d$. If $d \leq e$, then $\mathbf{x} \in S_e(\mathbf{x}) \cap S_e(\mathbf{y})$; so we may suppose that $d > e$.

Let $i_1, \ldots, i_d \leq n$ be the coordinate positions in which $\mathbf{x}$ and $\mathbf{y}$ differ: $x_{i_j} \neq y_{i_j}$, for $j = 1, \ldots, d$. Define $\mathbf{z} = (z_1, \ldots, z_n)$ by $z_k = y_k$ if $k \notin \{i_1, \ldots, i_e\}$ and $z_k = x_k$ if $k \in \{i_1, \ldots, i_e\}$. Then $d_H(\mathbf{y}, \mathbf{z}) = e$ and $d_H(\mathbf{x}, \mathbf{z}) = d - e \leq e$. Thus $\mathbf{z} \in S_e(\mathbf{x}) \cap S_e(\mathbf{y})$. Therefore (2) implies (3). □

A code C that satisfies the three equivalent properties of Lemma 2.2.1 is called an *e-error-correcting code*. The lemma reveals one of the most pleasing aspects of coding theory by identifying concepts from three distinct and important areas. The first property is algorithmic, the second is geometric, and the third is linear algebraic. We can readily switch from one point of view to another in search of appropriate insight and methodology as the context requires.

e-error-correcting code

(2.2.2) PROBLEM. *Explain why the error detecting algorithm $\mathbf{SS}_0$ correctly detects all patterns of fewer than d_{min} symbol errors.*

(2.2.3) PROBLEM. *Let $f \geq e$. Prove that the following are equivalent for the code C in A^n:*

(1) *under $\mathbf{SS}_e$ any occurrence of e or fewer symbol errors will always be successfully corrected and no occurrence of f or fewer symbol errors will cause a decoder error;*

(2) *for all distinct $\mathbf{x}, \mathbf{y}$ in C, we have $S_f(\mathbf{x}) \cap S_e(\mathbf{y}) = \emptyset$;*

(3) *the minimum distance of C, $d_{min}(C)$, is at least $e + f + 1$.*

A code C that satisfies the three equivalent properties of the problem is called an e-error-correcting, f-error-detecting code.

e-error-correcting, f-error-detecting

(2.2.4) PROBLEM. *Consider an erasure channel, that is, a channel that erases certain symbols and leaves a '?' in their place but otherwise changes nothing. Explain why, using a code with minimum distance d on this channel, we can correct all patterns of up to $d-1$ symbol erasures. (In certain computer systems this observation is used to protect against hard disk crashes.)*

By Lemma 2.2.1, if we want to construct an e-error-correcting code, we must be careful to choose as codewords the centers of radius e spheres that are pairwise disjoint. We can think of this as packing spheres of radius e into the large box that is the entire codespace. From this point of view, it is clear that we will not be able to fit in any number of spheres whose total volume exceeds the volume of the box. This proves:

(2.2.5) THEOREM. *(SPHERE PACKING CONDITION.) If C is an e-error-correcting code in A^n, then*

$$|C| \cdot |S_e(*)| \leq |A^n| . \qquad \square$$

Combined with Problem 2.1.4, this gives:

(2.2.6) COROLLARY. *(SPHERE PACKING BOUND; HAMMING BOUND.) If C is a m-ary e-error-correcting code of length n, then*

$$|C| \leq m^n \Big/ \sum_{i=0}^{e} \binom{n}{i} (m-1)^i . \qquad \square$$

perfect e-error-correcting code

A code C that meets the sphere packing bound with equality is called a *perfect e-error-correcting code*. Equivalently, C is a perfect e-error-correcting code if and only if $\mathbf{SS}_e$ is a **MDD** algorithm. As examples we have the binary repetition codes of odd length. The $[7,4]$ Hamming code is a perfect 1-error-correcting code, as we shall see in Section 4.1.

(2.2.7) THEOREM. *(GILBERT-VARSHAMOV BOUND.) There exists an m-ary e-error-correcting code C of length n such that*

$$|C| \geq m^n \Big/ \sum_{i=0}^{2e} \binom{n}{i} (m-1)^i .$$

PROOF. The proof is by a "greedy algorithm" construction. Let the codespace be A^n. At Step 1 we begin with the code $C_1 = \{\mathbf{x}_1\}$, for any word $\mathbf{x}_1$. Then, for $i \geq 2$, we have:

Step i. Set $S_i = \bigcup_{j=1}^{i-1} S_{d-1}(\mathbf{x}_j)$.
If $S_i = A^n$, halt.
Otherwise choose a vector $\mathbf{x}_i$ in $A^n - S_i$;
set $C_i = C_{i-1} \cup \{\mathbf{x}_i\}$;
go to Step $i+1$.

At Step i, the code C_i has cardinality i and is designed to have minimum distance at least d. (As long as $d \leq n$ we can choose $\mathbf{x}_2$ at distance d from $\mathbf{x}_1$; so each C_i, for $i \geq 1$ has minimum distance exactly d.)

How soon does the algorithm halt? We argue as we did in proving the sphere packing condition. The set $S_i = \bigcup_{j=1}^{i-1} S_{d-1}(\mathbf{x}_j)$ will certainly be smaller than A^n if the spheres around the words of C_{i-1} have total volume less than the volume of the entire space A^n; that is, if

$$|C_{i-1}| \cdot |S_{d-1}(*)| < |A^n| .$$

Therefore when the algorithm halts, this inequality must be false. Now Problem 2.1.4 gives the bound. □

A sharper version of the Gilbert-Varshamov bound exists, but the asymptotic result of the next section is unaffected.

EXAMPLES.

(i) Consider a binary 2-error-correcting code of length 90. By the Sphere Packing Bound it has size at most

$$\frac{2^{90}}{|S_2(*)|} = \frac{2^{90}}{2^{12}} = 2^{78} .$$

If a code existed meeting this bound, it would be perfect.

By the Gilbert-Varshamov Bound, in $\{0,1\}^{90}$ there exists a code C with minimum distance 5, which therefore corrects 2 errors, and having

$$|C| \geq \frac{2^{90}}{|S_4(*)|} = \frac{2^{90}}{2676766} \approx 4.62 \times 10^{20} .$$

As $2^{78} \approx 3.02 \times 10^{23}$, there is a factor of roughly 650 separating the lower and upper bounds.

(ii) Consider a ternary 2-error-correcting code of length 8. By the Sphere Packing Bound it has size bounded above by

$$\frac{3^8}{|S_2(*)|} = \frac{6561}{129} \approx 50.86 .$$

Therefore it has size at most $\lfloor 50.86 \rfloor = 50$. On the other hand, the Gilbert-Varshamov Bound guarantees only a code C of size bounded below by

$$|C| \geq \frac{6561}{|S_4(*)|} = \frac{6561}{1697} \approx 3.87 ,$$

that is, of size at least $\lceil 3.87 \rceil = 4$! Later we shall construct an appropriate C of size 27. (This is in fact the largest possible.)

(2.2.8) PROBLEM. *In each of the following cases decide whether or not there exists a 1-error-correcting code C with the given size in the codespace V. If there is such a code, give an example (except in (d), where an example is not required but a justification is). If there is not such a code, prove it.*

(a) $V = \{0,1\}^5$ *and* $|C| = 6$*;*
(b) $V = \{0,1\}^6$ *and* $|C| = 9$*;*
(c) $V = \{0,1,2\}^4$ *and* $|C| = 9$.
(d) $V = \{0,1,2\}^8$ *and* $|C| = 51$.

(2.2.9) PROBLEM. *In each of the following cases decide whether or not there exists a 2-error-correcting code C with the given size in the codespace V. If there is such a code, give an example. If there is not such a code, prove it.*

(*a*) $V = \{0,1\}^8$ *and* $|C| = 4$;

(*b*) $V = \{0,1\}^8$ *and* $|C| = 5$.

2.3 Shannon's theorem and the code region

The present section is devoted to information theory rather than coding theory and will not contain complete proofs. The goal of coding theory is to live up to the promises of information theory. Here we shall see of what our dreams are made.

Our immediate goal is to quantify the Fundamental Problem. We need to evaluate information content and error performance.

dimension

We first consider information content. The m-ary code C has *dimension* $k(C) = \log_m(|C|)$. The integer $k = \lceil k(C) \rceil$ is the smallest such that each message for C can be assigned its own individual message k-tuple from the m-ary alphabet A. Therefore we can think of the dimension as the number of codeword symbols that are carrying message rather than redundancy. (Thus

redundancy

the number $n - k$ is sometimes called the *redundancy* of C.) A repetition code has n symbols, only one of which carries the message; so its dimension is 1. For a length n parity check code, $n - 1$ of the symbols are message symbols; and so the code has dimension $n - 1$. The $[7, 4]$ Hamming code has dimension 4 as does its $[8, 4]$ extension, since both contain $2^4 = 16$ codewords. Our definition of dimension does not apply to our real Reed-Solomon example 1.3.6 since its alphabet is infinite, but it is clear what its dimension should be. Its 27 positions are determined by 7 free parameters, so the code should have dimension 7.

The dimension of a code is a deceptive gauge of information content. For instance, a binary code C of length 4 with 4 codewords and dimension $\log_2(4) = 2$ actually contains more information than a second code D of length 8 with 8 codewords and dimension $\log_2(8) = 3$. Indeed the code C can be used to produce $16 = 4 \times 4$ different valid code sequences of length 8 (a pair of codewords) while the code D only offers 8 valid sequences of length 8. Here and elsewhere, the proper measure of information content should be the fraction of the code symbols that carries information rather than redundancy. In this example $2/4 = 1/2$ of the symbols of C carry information while for D only $3/8$ of the symbols carry information, a fraction smaller than that for C.

The fraction of a repetition codeword that is information is $1/n$, and for a parity check code the fraction is $(n-1)/n$. In general, we define the *normalized*

rate

dimension or *rate* $\kappa(C)$ of the m-ary code C of length n by

$$\kappa(C) = k(C)/n = n^{-1} \log_m(|C|) .$$

The repetition code thus has rate $1/n$, and the parity check code rate $(n-1)/n$. The $[7, 4]$ Hamming code has rate $4/7$, and its extension rate $4/8 = 1/2$. The $[4, 2]$ ternary Hamming code has rate $2/4 = 1/2$. Our definition of rate does

not apply to the real Reed-Solomon example of 1.3.6, but arguing as before we see that it has "rate" 7/27. The rate is the normalized dimension of the code, in that it indicates the fraction of each code coordinate that is information as opposed to redundancy.

The rate $\kappa(C)$ provides us with a good measure of the information content of C. Next we wish to measure the error handling ability of the code. One possible gauge is $\mathcal{P}_C$, the error expectation of C; but in general this will be hard to calculate. We can estimate $\mathcal{P}_C$, for an $mSC(p)$ with small p, by making use of the obvious relationship $\mathcal{P}_C \leq \mathcal{P}_C(\mathbf{SS}_\rho)$ for any ρ. If $e = \lfloor (d-1)/2 \rfloor$, then C is an e-error-correcting code; and certainly $\mathcal{P}_C \leq \mathcal{P}_C(\mathbf{SS}_e)$, a probability that is easy to calculate. Indeed $\mathbf{SS}_e$ corrects all possible patterns of at most e symbol errors but does not correct any other errors; so

$$\mathcal{P}_C(\mathbf{SS}_e) = 1 - \sum_{i=0}^{e} \binom{n}{i} (m-1)^i p^i q^{n-i} .$$

The difference between $\mathcal{P}_C$ and $\mathcal{P}_C(\mathbf{SS}_e)$ will be given by further terms $p^j q^{n-j}$ with j larger than e. For small p, these new terms will be relatively small.

Shannon's theorem guarantees the existence of large families of codes for which $\mathcal{P}_C$ is small. The previous paragraph suggests that to prove this efficiently we might look for codes with arbitrarily small $\mathcal{P}_C(\mathbf{SS}_{(d_{min}-1)/2})$, and in a sense we do. However, it can be proven that decoding up to minimum distance alone is not good enough to prove Shannon's Theorem. (Think of the 'Birthday Paradox'.) Instead we note that a received block of large length n is most likely to contain sn symbol errors where $s = p(m-1)$ is the probability of symbol error. Therefore in proving Shannon's theorem we look at large numbers of codes, each of which we decode using $\mathbf{SS}_\rho$ for some radius ρ a little larger than sn.

A family $\mathcal{C}$ of codes over A is called a *Shannon family* if, for every $\epsilon > 0$, there is a code $C \in \mathcal{C}$ with $\mathcal{P}_C < \epsilon$. For a finite alphabet A, the family $\mathcal{C}$ must necessarily be infinite and so contain codes of unbounded length. Shannon family

(2.3.1) PROBLEM. *Prove that the set of all binary repetition codes of odd length is a Shannon family on $BSC(p)$ for $p < 1/2$.*

Although repetition codes give us a Shannon family, they do not respond to the Fundamental Problem by having good information content as well. Shannon proved that codes of the sort we need are out there somewhere.

(2.3.2) THEOREM. *(*SHANNON'S CHANNEL CODING THEOREM.*) Consider the m-ary symmetric channel $mSC(p)$, with $p < 1/m$. There is a function $C_m(p)$ such that, for any $\kappa < C_m(p)$,*

$$\mathcal{C}_\kappa = \{ \text{ } m\text{-ary block codes of rate at least } \kappa \}$$

is a Shannon family. Conversely if $\kappa > C_m(p)$, then $\mathcal{C}_\kappa$ is not a Shannon family. □

The function $C_m(p)$ is the capacity function for the $mSC(p)$ and will be discussed below.

Shannon's theorem tells us that we can communicate reliably at high rates; but, as R.J. McEliece has remarked, its lesson is deeper and more precise than this. It tells us that to make the best use of our channel we must transmit at rates near capacity and then filter out errors at the destination. Think about Lucy and Ethel wrapping chocolates. The company may maximize its total profit by increasing the conveyor belt rate and accepting a certain amount of wastage. The tricky part is figuring out how high the rate can be set before chaos ensues.

Shannon's theorem is robust in that bounding rate by the capacity function still allows transmission at high rate for most p. In the particular case $m = 2$, we have

$$C_2(p) = 1 + p\log_2(p) + q\log_2(q)\,,$$

where $p+q = 1$. Thus on a binary symmetric channel with transition probability $p = .02$ (a pretty bad channel), we have $C_2(.02) \approx .8586$. Similarly $C_2(.1) \approx .5310$, $C_2(.01) \approx .9192$, and $C_2(.001) \approx .9886$. So, for instance, if we expect bit errors .1 % of the time, then we may transmit messages that are nearly 99% information but still can be decoded with arbitrary precision. Many channels in use these days operate with p between 10^{-7} and 10^{-15}.

We define the general entropy and capacity functions before giving an idea
entropy of their origin. The m-ary *entropy* function is defined on $(0, (m-1)/m]$ by

$$H_m(x) = -x\log_m(x/(m-1)) - (1-x)\log_m(1-x),$$

where we additionally define $H_m(0) = 0$ for continuity. Notice $H_m(\frac{m-1}{m}) =$
capacity 1. Having defined entropy, we can now define the m-ary *capacity* function on $[0, 1/m]$ by

$$C_m(p) = 1 - H_m((m-1)p)\,.$$

We have $C_m(0) = 1$ and $C_m(1/m) = 0$.

We next see why entropy and capacity might play a role in coding problems. (The lemma is a consequence of Stirling's formula.)

(2.3.3) LEMMA. *For spheres in A^n with $|A| = m$ and any σ in $(0, (m-1)/m]$, we have*

$$\lim_{n\to\infty} n^{-1}\log_m(|S_{\sigma n}(*)|) = H_m(\sigma). \qquad \square$$

For a code C of sufficient length n on $mSC(p)$ we expect sn symbol errors in a received word, so we would like to correct at least this many errors. Applying the Sphere Packing Condition 2.2.5 we have

$$|C| \cdot |S_{sn}(*)| \le m^n\,,$$

which, upon taking logarithms, is

$$\log_m(|C|) + \log_m(|S_{sn}(*)|) \le n\,.$$

We divide by n and move the second term across the inequality to find

$$\kappa(C) = n^{-1}\log_m(|C|) \leq 1 - n^{-1}\log_m(|S_{sn}(*)|)\,.$$

The righthand side approaches $1 - H_m(s) = C_m(p)$ as n goes to infinity; so, for C to be a contributing member of a Shannon family, it should have rate at most capacity. This suggests:

(2.3.4) PROPOSITION. *If $\mathcal{C}$ is a Shannon family for $mSC(p)$ with $0 \leq p \leq 1/m$, then* $\liminf_{C\in\mathcal{C}} \kappa(C) \leq C_m(p)$. □

The proposition provides the converse in Shannon's Theorem, as we have stated it. (Our arguments do not actually prove this converse. We can not assume our spheres of radius sn to be pairwise disjoint, so the Sphere Packing Condition does not directly apply.)

We next suggest a proof of the direct part of Shannon's theorem, noticing along the way how our geometric interpretation of entropy and capacity is involved.

The outline for a proof of Shannon's theorem is short: for each $\epsilon > 0$ (and n) we choose a $\rho\,(= \rho(\epsilon, n))$ for which

$$\mathrm{avg}_C\ \mathcal{P}_C(\mathbf{SS}_\rho) < \epsilon\,,$$

for all sufficiently large n, where the average is taken over all $C \subseteq A^n$ with $|C| = m^{\kappa n}$ (round up), codes of length n and rate κ. As the average is less than ϵ, there is certainly some particular code C with $\mathcal{P}_C$ less than ϵ, as required.

In carrying this out it is enough (by symmetry) to consider all C containing a fixed $\mathbf{x}$ and prove

$$\mathrm{avg}_C\ \mathcal{P}_{\mathbf{x}}(\mathbf{SS}_\rho) < \epsilon\,.$$

Two sources of incorrect decoding for transmitted $\mathbf{x}$ must be considered:

(i) $\mathbf{y}$ is received with $\mathbf{y} \notin S_\rho(\mathbf{x})$;

(ii) $\mathbf{y}$ is received with $\mathbf{y} \in S_\rho(\mathbf{x})$ but also $\mathbf{y} \in S_\rho(\mathbf{z})$, for some $\mathbf{z} \in C$ with $\mathbf{z} \neq \mathbf{x}$.

For mistakes of the first type the binomial distribution guarantees a probability less than $\epsilon/2$ for a choice of ρ just slightly larger than $sn = p(m-1)n$, even without averaging. For our fixed $\mathbf{x}$, the average probability of an error of the second type is over-estimated by

$$m^{\kappa n}\frac{|S_\rho(\mathbf{z})|}{m^n}\,,$$

the number of $\mathbf{z} \in C$ times the probability that an arbitrary $\mathbf{y}$ is in $S_\rho(\mathbf{z})$. This average probability has logarithm

$$-n\Big((1 - n^{-1}\log_m(|S_\rho(*)|)) - \kappa\Big)\,.$$

In the limit, the quantity in the parenthesis is

$$(1 - H_m(s)) - \kappa = \beta\,,$$

which is positive by hypothesis. The average then behaves like $m^{-n\beta}$. Therefore by increasing n we can also make the average probability in the second case less than $\epsilon/2$. This completes the proof sketch.

Shannon's theorem now guarantees us codes with arbitrarily small error expectation $\mathcal{P}_C$, but this number is still not a very good measure of error handling ability for the Fundamental Problem. Aside from being difficult to calculate, it is actually channel dependent, being typically a polynomial in p and $q = 1 - (m-1)p$. As we have discussed, one of the attractions of **IMDD** decoding on m-ary symmetric channels is the ability to drop channel specific parameters in favor of general characteristics of the code geometry. So perhaps rather than search for codes with small $\mathcal{P}_C$, we should be looking at codes with large minimum distance. This parameter is certainly channel independent; but, as with dimension and rate, we have to be careful to normalize the distance. While 100 might be considered a large minimum distance for a code of length 200, it might not be for a code of length 1,000,000. We instead consider the *normalized distance* of the length n code C defined as $\delta(C) = d_{min}(C)/n$.

normalized distance

As further motivation for study of the normalized distance, we return to the observation that, in a received word of decent length n, we expect $p(m-1)n$ symbol errors. For correct decoding we would like

$$p(m-1)n \leq (d_{min} - 1)/2\,.$$

If we rewrite this as

$$0 < 2p(m-1) \leq (d_{min} - 1)/n < d_{min}/n = \delta\,,$$

then we see that for a family of codes with good error handling ability we attempt to bound the normalized distance δ away from 0.

The Fundamental Problem has now become:

> **The Fundamental Problem of Coding Theory** — Find practical m-ary codes C with reasonably large rate $\kappa(C)$ and reasonably large normalized distance $\delta(C)$.

What is viewed as practical will vary with the situation. For instance, we might wish to bound decoding complexity or storage required.

Shannon's theorem provides us with cold comfort. The codes are out there somewhere, but the proof by averaging gives no hint as to where we should look.[2] In the next chapter we begin our search in earnest. But first we discuss what sort of pairs $(\delta(C), \kappa(C))$ we might attain.

[2]In the last fifty years many good codes have been constructed; but only beginning in 1993, with the introduction of "turbo codes" and the intense study of related codes and associated iterative decoding algorithms, did we start to see how Shannon's bound might be approachable in practice in certain cases. These notes do not address such recent topics. The codes and algorithms discussed here remain of importance. The newer constructions are not readily adapted to things like compact discs, computer memories, and other channels somewhat removed from those of Shannon's theorem.

We could graph in $[0,1]\times[0,1]$ all pairs $(\delta(C),\kappa(C))$ realized by some m-ary code C, but many of these correspond to codes that have no claim to being practical. For instance, the length 1 binary code $C=\{0,1\}$ has $(\delta(C),\kappa(C))=(1,1)$ but is certainly impractical by any yardstick. The problem is that in order for us to be confident that the number of symbol errors in a received n-tuple is close to $p(m-1)n$, the length n must be large. So rather than graph all attainable pairs $(\delta(C),\kappa(C))$, we adopt the other extreme and consider only those pairs that can be realized by codes of arbitrarily large length.

To be precise, the point $(\delta,\kappa)\in[0,1]\times[0,1]$ belongs to the m-ary *code region* if and only if there is a sequence $\{C_n\}$ of m-ary codes C_n with unbounded length n for which

code region

$$\delta=\lim_{n\to\infty}\delta(C_n) \text{ and } \kappa=\lim_{n\to\infty}\kappa(C_n)\ .$$

Equivalently, the code region is the set of all accumulation points in $[0,1]\times[0,1]$ of the graph of achievable pairs $(\delta(C),\kappa(C))$.

(2.3.5) THEOREM. *(*MANIN'S BOUND ON THE CODE REGION.*) There is a continuous, nonincreasing function $\alpha_m(\delta)$ on the interval $[0,1]$ such that the point (δ,κ) is in the m-ary code region if and only if*

$$0\le\kappa\le\alpha_m(\delta)\,. \qquad \square$$

Although the proof is elementary, we do not give it. However we can easily see why something like this should be true. If the point (δ,κ) is in the code region, then it seems reasonable that the code region should contain as well the points (δ',κ) , $\delta'<\delta$, corresponding to codes with the same rate but smaller distance and also the points (δ,κ'), $\kappa'<\kappa$, corresponding to codes with the same distance but smaller rate. Thus for any point (δ,κ) of the code region, the rectangle with corners $(0,0)$, $(\delta,0)$, $(0,\kappa)$, and (δ,κ) should be entirely contained within the code region. Any region with this property has its upper boundary function nonincreasing and continuous.

In our discussion of Proposition 2.3.4 we saw that $\kappa(C)\le 1-H_m(s)$ when correcting the expected sn symbol errors for a code of length n. Here sn is roughly $(d-1)/2$ and s is approximately $(d-1)/2n$. In the present context the argument preceding Proposition 2.3.4 leads to

(2.3.6) THEOREM. *(*ASYMPTOTIC HAMMING BOUND.*) We have*

$$\alpha_m(\delta)\le 1-H_m(\delta/2)\ . \qquad \square$$

Similarly, from the Gilbert-Varshamov bound 2.2.7 we derive:

(2.3.7) THEOREM. *(*ASYMPTOTIC GILBERT-VARSHAMOV BOUND.*) We have*

$$\alpha_m(\delta)\ge 1-H_m(\delta)\ . \qquad \square$$

Various improvements to the Hamming upper bound and its asymptotic version exist. We present two.

(2.3.8) THEOREM. *(PLOTKIN BOUND.) Let C be an m-ary code of length n with $\delta(C) > (m-1)/m$. Then*

$$|C| \leq \frac{\delta}{\delta - \frac{m-1}{m}} \ . \qquad \square$$

(2.3.9) COROLLARY. *(ASYMPTOTIC PLOTKIN BOUND.)*
(1) *$\alpha_m(\delta) = 0$ for $(m-1)/m < \delta \leq 1$.*
(2) *$\alpha_m(\delta) \leq 1 - \frac{m}{m-1}\delta$ for $0 \leq \delta \leq (m-1)/m$.* $\square$

For a fixed $\delta > (m-1)/m$, the Plotkin bound 2.3.8 says that code size is bounded by a constant. Thus as n goes to infinity, the rate goes to 0, hence (1) of the corollary. Part (2) is proven by applying the Plotkin bound not to the code C but to a related code C' with the same minimum distance but of shorter length. (The proof of part (2) of the corollary appears below in §6.1.3. The proof of the theorem is given as Problem 3.1.6.)

(2.3.10) PROBLEM. *(SINGLETON BOUND.) Let C be a code in A^n with minimum distance $d = d_{min}(C)$. Prove $|C| \leq |A|^{n-d+1}$. (HINT: For the word $\mathbf{y} \in A^{n-d+1}$, how many codewords of C can have a copy of $\mathbf{y}$ as their first $n-d+1$ entries?)*

(2.3.11) PROBLEM. *(ASYMPTOTIC SINGLETON BOUND.) Use Problem 2.3.10 to prove $\alpha_m(\delta) \leq 1 - \delta$. (We remark that this is a weak form of the asymptotic Plotkin bound.)*

While the asymptotic Gilbert-Varshamov bound shows that the code region is large, the proof is essentially nonconstructive since the greedy algorithm must be used infinitely often. Most of the easily constructed families of codes give rise to code region points either on the δ-axis or the κ-axis.

(2.3.12) PROBLEM. *Prove that the family of repetition codes produces the point $(1,0)$ of the code region and the family of parity check codes produces the point $(0,1)$.*

The first case in which points in the interior of the code region were explicitly constructed was the following 1972 result of Justesen:

(2.3.13) THEOREM. *For $0 < \kappa < \frac{1}{2}$, there is a positive constant c and a sequence of binary codes $J_{\kappa,n}$ with rate at least κ and*

$$lim_{n\to\infty}\delta(J_{\kappa,n}) \geq c(1-2\kappa)\,.$$

Thus the line $\delta = c(1-2\kappa)$ is constructively within the binary code region. $\square$

Justesen also has a version of his construction that produces binary codes of larger rate. The constant c that appears in Theorem 2.3.13 is the unique solution to $H_2(c) = \frac{1}{2}$ in $[0, \frac{1}{2}]$ and is roughly .110.

While there are various improvements to the asymptotic Hamming upper bound on $\alpha_m(\delta)$ and the code region, such improvements to the asymptotic Gilbert-Varshamov lower bound are rare and difficult. Indeed for a long time

Figure 2.1: Bounds on the m-ary code region

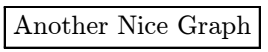


Figure 2.2: The 49-ary code region

it was conjectured that the asymptotic Gilbert-Varshamov bound holds with equality,

$$\alpha_m(\delta) = 1 - H_m(\delta)\,.$$

This is now known to be false for infinitely many m, although not as yet for the important cases $m = 2, 3$. The smallest known counterexample is at $m = 49$.

(2.3.14) THEOREM. *The line*

$$\kappa + \delta = \frac{5}{6}$$

is within the 49*-ary code region but is not below the corresponding Gilbert-Varshamov curve*

$$\kappa = 1 - H_{49}(\delta)\,. \qquad \square$$

This theorem and much more was proven by Tsfasman, Vladut, and Zink in 1982 using difficult results from algebraic geometry in the context of a broad generalization of Reed-Solomon codes.

It should be emphasized that these results are of an asymptotic nature. As we proceed, we shall see various useful codes for which (δ, κ) is outside the code region and important families whose corresponding limit points lie on a coordinate axis $\kappa = 0$ or $\delta = 0$.

Chapter 3

Linear Codes

In order to define codes that we can encode and decode efficiently, we add more structure to the codespace. We shall be mainly interested in linear codes. A *linear code* of length n over the field F is a subspace of F^n. Thus the words of the codespace F^n are vectors, and we often refer to codewords as *codevectors*.

linear code

codevectors

In the first section we develop the basics of linear codes, in particular we introduce the crucial concept of the dual of a code. The second and third sections then discuss the general principles behind encoding and decoding linear codes. We encounter the important concept of a syndrome.

3.1 Basics

If C is a linear code that, as a vector space over the field F, has dimension k, then we say that C is an $[n, k]$ *linear code* over F, or an $[n, k]$ code, for short. There is no conflict with our definition of the dimension of C as a code, since $|C| = |F|^k$. (Indeed the choice of general terminology was motivated by the special case of linear codes.) In particular the rate of an $[n, k]$ linear code is k/n. If C has minimum distance d, then C is an $[n, k, d]$ linear code over F. The number $n - k$ is again the *redundancy* of C.

$[n, k]$ linear code

redundancy

We begin to use $\mathbb{F}_2$ in preference to $\{0, 1\}$ to denote our binary alphabet, since we wish to emphasize that the alphabet carries with it an arithmetic structure. Similar remarks apply to ternary codes.

EXAMPLES. (i) The repetition code of length n over F is an $[n, 1, n]$ linear code.

(ii) The binary parity check code of length n is an $[n, n-1, 2]$ linear code.

(iii) The $[7, 4]$, $[8, 4]$, and $[4, 2]$ Hamming codes of the introduction were all defined by parity considerations or similar equations. We shall see below that this forces them to be linear.

(iv) The real Reed-Solomon code of our example is a $[27, 7, 21]$ linear code over the real numbers $\mathbb{R}$.

(3.1.1) THEOREM. *(SHANNON'S THEOREM FOR LINEAR CODES.) Let F be a field with m elements, and consider a $mSC(p)$ with $p < 1/m$. Set*

$$\mathcal{L}_\kappa = \{ \text{ linear codes over } F \text{ with rate at least } \kappa \}.$$

Then $\mathcal{L}_\kappa$ is a Shannon family provided $\kappa < C_m(p)$. □

Forney (1966) proved a strong version of this theorem which says that we need only consider those linear codes of length n with encoder/decoder complexity on the order of n^4 (but at the expense of using very long codes). Thus there are Shannon families whose members have rate approaching capacity and are, in a theoretical sense, practical[1].

Hamming weight

minimum weight

The *Hamming weight* (for short, *weight*) of a vector $\mathbf{v}$ is the number of its nonzero entries and is denoted $w_H(\mathbf{v})$. We have $w_H(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0})$. The *minimum weight* of the code C is the minimum nonzero weight among all codewords of C,

$$w_{min}(C) = w(C) = \min_{\mathbf{0} \neq \mathbf{x} \in C} (w_H(\mathbf{x})) .$$

(3.1.2) LEMMA. *Over a field, Hamming distance is translation invariant. In particular, for linear codes, the minimum weight equals the minimum distance.*

PROOF. Clearly $d_H(\mathbf{x}, \mathbf{y}) = d_H(\mathbf{x} - \mathbf{z}, \mathbf{y} - \mathbf{z})$ for all $\mathbf{z}$. In particular

$$d_H(\mathbf{x}, \mathbf{y}) = d_H(\mathbf{x} - \mathbf{y}, \mathbf{y} - \mathbf{y}) = d_H(\mathbf{x} - \mathbf{y}, \mathbf{0}) . \qquad \square$$

A consequence of the lemma is that minimum distance for linear codes is much easier to calculate than for arbitrary codes. One need only survey $|C|$ codewords for weight rather than roughly $|C|^2$ pairs for distance.

EXAMPLES. Of course the minimum weight of the length n repetition code is n. Also the minimum weight of the parity check code is clearly 2. The minimum weight of the length 27 real Reed-Solomon code is equal to its minimum distance which we found to be 21. We listed the codewords of the $[4, 2]$ ternary Hamming code, and so it visibly has minimum weight 3.

Verifying that the minimum weight of the $[7, 4]$ Hamming code is 3 is easy to do directly by hand, but we will give a conceptual way of doing this calculation below. The extended $[8, 4]$ Hamming code adds an overall parity check bit to the $[7, 4]$ code, so its minimum weight is 4.

The following elementary property of binary weights can be very helpful. For instance, it proves directly that the parity check code is linear.

(3.1.3) PROBLEM. *Prove that, for binary vectors $\mathbf{x}$ and $\mathbf{y}$ of the same length, we have*

$$w_H(\mathbf{x} + \mathbf{y}) = w_H(\mathbf{x}) + w_H(\mathbf{y}) - 2w_H(\mathbf{x} * \mathbf{y})$$

where $\mathbf{x} * \mathbf{y}$ is defined to have a 1 only in those positions where both $\mathbf{x}$ and $\mathbf{y}$ have a 1.

The matrix G is a *spanning matrix* for the linear code C provided $C = \mathbf{RS}(G)$, the row space of G. A *generator matrix* of the $[n,k]$ linear code C over F is a $k \times n$ matrix G with $C = \mathbf{RS}(G)$. Thus a generator matrix is a spanning matrix whose rows are linearly independent. We may easily construct many codes using generator matrices. Of course it is not clear from the matrix how good the code will be.

spanning matrix
generator matrix

EXAMPLES. (i) The repetition code has generator matrix

$$G = \Big[1, 1, \ldots, 1\Big].$$

(ii) A particularly nice generator matrix for the parity check code is

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 1 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix},$$

composed of all weight 2 codewords with a one in the last column. This code will have many other generator matrices as well. Here are two for the $[7,6]$ parity check code:

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

(iii) Consider the $[7,4]$ Hamming code of Example 1.3.3. In turn we set the four message symbols (X_3, X_5, X_6, X_7) to $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, and $(0,0,0,1)$. The four resulting codewords form the rows of a generator matrix. We find

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

(iv) A generator matrix for the $[8,4]$ extended Hamming code of Example 1.3.4 results from adding a column to that for the $[7,4]$ code, each new entry checking parity of that row in the matrix. We have

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

[1] Oxymoron!

(v) For a generator matrix of the $[4,2]$ ternary Hamming code of Example 1.3.5, we may set (a,b) equal to $(1,0)$ and $(0,1)$ in turn to get the matrix

$$\begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix},$$

although any pair of codewords would do as rows provided one is not a multiple of the other. For instance

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 \end{bmatrix}$$

is also a generator matrix.

(3.1.4) PROBLEM. *Prove that, in a linear code over the field $\mathbb{F}_q$, either all of the codewords begin with 0 or exactly $1/q$ of the codewords begin with 0. (You might want first to consider the binary case.)*

(3.1.5) PROBLEM. *Let C be a linear $[n,k,d]$ code over the field $\mathbb{F}_q$.*

*(a) Prove that the sum of all the weights of all the codewords of C is at most $n(q-1)q^{k-1}$. (*HINT: *Use the previous problem.)*

*(b) Prove that the minimum distance d of C is at most $\dfrac{n(q-1)q^{k-1}}{q^k-1}$. (*HINT: *The minimum weight is less than or equal to the average nonzero weight.)*

(c) Prove the Plotkin bound for linear codes with $d/n > (q-1)/q$:

$$|C| \leq \frac{d}{d - \frac{q-1}{q}n} .$$

(3.1.6) PROBLEM. *Prove the Plotkin bound for a general m-ary code C of length n and minimum distance d with $d/n > (m-1)/m$:*

$$|C| \leq \frac{d}{d - \frac{m-1}{m}n} .$$

*(*HINT: *Find an upper bound on the average nonzero distance between codewords by comparing all distinct pairs of codewords and examining each coordinate position in turn.)*

dual code

Let C be any code (not necessarily linear) in F^n, for F a field. The *dual code* of C, denoted $C^\perp$, is the code

$$C^\perp = \{\mathbf{x} \in F^n \mid \mathbf{x} \cdot \mathbf{c} = 0, \text{ for all } \mathbf{c} \in C\},$$

where $\mathbf{x} \cdot \mathbf{c}$ is the usual dot product. The dual of C is linear even if C is not. (This is often a good way of proving that a given code is linear.) We can in turn examine the dual of the dual and discover easily that $(C^\perp)^\perp = C^{\perp\perp} \supseteq C$.

If C is itself a linear code, then in fact $C^{\perp\perp} = C$. For instance, the dual of the binary repetition code of length n is the parity check code of length n; and the dual of the parity check code of length n is the repetition code of length n. To see that $C^{\perp\perp} = C$ for linear C, we use another description of $C^\perp$. Let G be a generator matrix for C. Then $\mathbf{x}$ is in $C^\perp$ if and only if $G\mathbf{x}^\top = \mathbf{0}$. Thus

the vectors of $C^\perp$ are precisely the transposes of the vectors of the null space $\mathbf{NS}(G)$. Therefore by Theorem A.1.7 the dimension of C plus the dimension of $C^\perp$ equals the length n, that is, $C^\perp$ has dimension $n-k$. Calculating dimensions twice, we learn that $C^{\perp\perp}$ has dimension k. As this space contains C and has the same dimension as C, it is equal to C. In summary:

(3.1.7) LEMMA. *If C is an $[n,k]$ linear code over F, then its dual $C^\perp$ is an $[n,n-k]$ linear code over F and $C^{\perp\perp}=C$.* □

The linear code C is *self-orthogonal* if $C^\perp \geq C$ and is *self-dual* if $C^\perp = C$. So, for instance, a binary repetition code of even length is self-orthogonal, as is the $[7,3]$ binary dual Hamming code. Since the dimension of a code plus that of its dual add up to the length, a self-dual code must be a $[2k,k]$ linear code, for some k. The $[8,4]$ extended Hamming code is self-dual, as can be easily checked using the generator matrix given above. The ternary $[4,2]$ Hamming code is also self-dual, as is easily checked.

self-orthogonal

self-dual

A generator matrix H for the dual code $C^\perp$ of the linear C is sometimes called a *check matrix* for C. In general it is not difficult to calculate a check matrix for a code, given a generator matrix G. Indeed if we pass to a generator in $\mathcal{RREF}$, then it is easy to find a basis for the null space and so for $C^\perp$ by following the remarks of Section A.1.3 of the appendix. In particular, if the generator matrix G (or its $\mathcal{RREF}$) has the special form

check matrix

$$\left[\; I_{k\times k} \mid A_{k\times n-k} \;\right]$$

then one check matrix is

$$H = \left[-A^\top_{n-k\times k} \mid I_{n-k\times n-k}\;\right].$$

(3.1.8) PROBLEM. *Consider a binary code of length* 16 *written as* 4×4 *square matrices. The code E is composed of every 4×4 binary matrix M such that:*
(*i*) *every row of M contains an even number of* 1*'s; and*
(*ii*) either *every column of M contains an even number of* 1*'s* or *every column of M contains an odd number of* 1*'s.*

(*a*) *Prove that E is a linear code.*
(*b*) *What is the dimension of E?*
(*c*) *What is the minimum distance of E?*
(*d*) *If the matrix*

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

is received, give all possible decodings subject to **MDD**. *That is, find all code matrices in E that are at minimum distance from this matrix.*

(3.1.9) PROBLEM. *Consider a binary code of length* 21 *whose words are written as arrays in the following gem shape:*

$$\begin{array}{ccccc} & x_1 & x_2 & x_3 & \\ x_4 & x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} & x_{13} \\ x_{14} & x_{15} & x_{16} & x_{17} & x_{18} \\ & x_{19} & x_{20} & x_{21} & \end{array}$$

The code E is composed of every binary array M of this shape and such that:
(i) every row of M contains an even number of 1*'s; and*
(ii) every column of M contains an even number of 1*'s.*

(a) Prove that E is a linear code.
(b) What is the dimension of E?
(c) What is the minimum distance of E?
(d) If the array

$$\begin{array}{ccccc} & 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 0 & \end{array}$$

is received, give all possible decodings subject to **MDD**. *That is, find all codewords in E that are closest to this array.*

(e) If the array

$$\begin{array}{ccccc} & 1 & 0 & 1 & \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ & 1 & 0 & 1 & \end{array}$$

is received, give all possible decodings subject to **MDD**.

(3.1.10) PROBLEM. *If C is a binary $[n, k]$ linear code, prove that either all weights of codewords of C are even or the even weight codewords of C form a linear $[n, k-1]$ subcode B. In the second case, how can the dual code of B be constructed from the dual code of C?*

(3.1.11) PROBLEM. *(a) Let C be a self-orthogonal binary linear code. Prove that all of its codewords have even weight. If additionally C has a spanning set composed of codewords with weights a multiple of* 4*, prove that every codeword has weight a multiple of* 4*.*

(b) Prove that a linear ternary code is self-orthogonal if and only if all its weights are a multiple of three.

If C is a binary code and $\mathbf{x}$ is a vector of $C^{\perp}$ then $\mathbf{c}\cdot\mathbf{x} = 0$, for all $\mathbf{c} \in C$; so $\mathbf{x}$ can be thought of as checking the parity of a subset of the coordinate positions of C, those positions in which $\mathbf{x}$ equals one. Extending this idea to nonbinary linear codes, we consider any vector of the dual as providing the coefficients of a "parity check equation" on the entries of codewords. The rows of a check matrix provide a basis for the space of parity check equations satisfied by the code, hence the terminology.

Because $C^{\perp\perp} = C$, we can use a check matrix H for C to give a concise definition of C:

$$C = \{\, \mathbf{x} \mid H\mathbf{x}^\top = \mathbf{0} \,\}.$$

Any matrix H for which $C = \{\, \mathbf{x} \mid H\mathbf{x}^\top = \mathbf{0} \,\}$ we shall call a *control matrix* for C. (This terminology is not common.) Thus a check matrix is a special kind of control matrix. A check matrix must have linearly independent rows while a control matrix need not.

control matrix

We often define a code in terms of a check matrix (or control matrix). In Example 1.3.5 we defined the $[4, 2]$ ternary Hamming code to be all 4-tuples (a, b, c, d) from $\{0, 1, 2\}^4$ that satisfy $a + b = c$ and $b + c + d = 0$. That is, we defined the code via the check matrix

$$\begin{bmatrix} 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Here the first check row requires that, for (a, b, c, d) to be in the code,

$$(a, b, c, d) \cdot (1, 1, 2, 0) = a + b + 2c = 0\,,$$

that is, $a + b = c$; and the second forces $b + c + d = 0$.

Shannon's discussion under Examples 1.3.3 of the $[7, 4]$ binary Hamming code essentially defines the code by its check matrix

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

Here every nonzero binary 3-tuple occurs exactly once as a column of the check matrix. The columns have been arranged so that column i is the binary representation of the integer i.

What is the minimum weight of the $[7, 4]$ Hamming code? If $\mathbf{x}$ is a vector of weight 1, then the product $H\mathbf{x}^\top$ is a column of H, indeed column i of H if the single 1 of $\mathbf{x}$ is in position i. As all columns of H are nonzero, $H\mathbf{x}^\top$ is also nonzero; so $\mathbf{x}$ is not a codeword. If instead $\mathbf{x}$ has weight 2, then $H\mathbf{x}^\top$ is the sum of two columns of H, those columns in which $\mathbf{x}$ equals 1. As no two columns are equal, this sum is never $\mathbf{0}$; so again $\mathbf{x}$ is not a codeword. On the other hand, it is possible to find three columns of H that sum to $\mathbf{0}$ (for instance, the first three); so the code does contain words of weight 3 (for instance, $(1, 1, 1, 0, 0, 0, 0)$). Therefore this code has minimum weight 3.

It is not difficult to generalize these arguments. Block matrix multiplication implies that, for any matrix H and row vector $\mathbf{x}$, the matrix product $H\mathbf{x}^\top$ is a linear combination of the columns of H with coefficients provided by $\mathbf{x}$, namely the sum $\sum_i \mathbf{h}_i x_i$ where H has i^{th} column $\mathbf{h}_i$ and x_i is the i^{th} entry of $\mathbf{x}$. In particular the entries of a nonzero codeword $\mathbf{x}$ give the coefficients of a linear dependence among the columns of H, a check matrix (or control matrix). Of course any column $\mathbf{h}_i$ that is multiplied by a scalar $x_i = 0$ makes no contribution to this linear combination. The nonzero entries of the codeword are the coefficients of a linear dependence among only those columns $\mathbf{h}_i$ for which the coefficient x_i is not 0. We are led to:

(3.1.12) LEMMA. *Let C be a linear code with control matrix H. A set of w columns of H is linearly dependent if and only if there is a nonzero codeword in C all of whose nonzero entries occur among coordinate positions corresponding to members of that column set. In particular $d_{min}(C) = d$ if and only if there exists a set of d linearly dependent columns in H but no set of $d-1$ linearly dependent columns.*

PROOF. All but the last sentence was discussed above. By Lemma 3.1.2 $d_{min}(C) = w_{min}(C)$. Now $w_{min}(C) \leq d$ if and only if there are d linearly dependent columns in H, while $w_{min}(C) \geq d$ if and only if all collections of $d-1$ columns are linearly independent. □

(3.1.13) PROBLEM. *Let*

$$H = \left[\mathbf{h}_1,\ \mathbf{h}_2,\ \ldots\ ,\ \mathbf{h}_n \right]$$

be the check matrix of the e-error-correcting, binary $[n,k]$ linear code D, the various $\mathbf{h}_j$ being the columns of H. Next let D' be the binary $[n,k]$ linear code with check matrix

$$H' = \left[\mathbf{h}_1,\ \mathbf{h}_1+\mathbf{h}_2,\ \mathbf{h}_1+\mathbf{h}_3,\ \ldots\ ,\ \mathbf{h}_1+\mathbf{h}_n \right] .$$

Prove that D' is also an e-error-correcting code.

(3.1.14) THEOREM. *(*SINGLETON BOUND.*) If C is an $[n,k]$ linear code over the field F, then*

$$d_{min}(C) \leq n-k+1 .$$

PROOF. Every $n-k \times n-k+1$ submatrix of a check matrix has rank at most $n-k$, so every set of $n-k+1$ columns of the check matrix is linearly dependent. The theorem then follows from Lemma 3.1.12. □

We have seen in Problem 2.3.10 that this result is true even for nonlinear codes. Indeed if we move k and $d = d_{min}(C)$ to opposite sides and raise $q = |F|$ to the appropriate power, we are left with

$$|C| = q^k \leq q^{n-d+1} .$$

The present proof of the bound shows that even more is true. Any set of $n-k+1$ coordinate positions contains the support (the nonzero entries) of a nonzero codeword.

(3.1.15) PROBLEM. *Use a generator matrix in $\mathcal{RREF}$ to give another quick proof of the Singleton bound for linear codes.*

maximum distance separable
MDS code

A linear code that meets the Singleton bound with equality is called *maximum distance separable* or, for short, an *MDS code*. Every subset of $n-k+1$ coordinate positions supports a codeword in an MDS code. By convention the zero code $\{\mathbf{0}\}$ is MDS, even though its minimum distance is somewhat ill-defined.

The $[4, 2]$ ternary Hamming code has minimum distance 3 and so is MDS since $3 = 4 - 2 + 1$. We shall meet many MDS codes later when we discuss the generalized Reed-Solomon codes.

(3.1.16) PROBLEM. *Prove that the dual of an MDS codes is also an MDS code.*

(3.1.17) PROBLEM. *Prove that a binary MDS code of length n is one of $\{\mathbf{0}\}$, the repetition code, the parity check code, or all $\mathbb{F}_2^n$.*

3.2 Encoding and information

If we are transmitting with an $[n, k]$ linear code over the field F, then we think of our message as being provided as k-tuples from F, members of the space F^k. We can encode using the generator matrix G by mapping the message k-tuple $\mathbf{x}$ to the codeword $\mathbf{x}G$. Here $\mathbf{x}G$ is a codeword since, by matrix block multiplication, it is a linear combination of the rows of G (with coefficients given by $\mathbf{x}$) and $C = \mathbf{RS}(G)$.

The $k \times n$ generator matrix G is a *standard generator matrix* if its first k columns form a $k \times k$ identity matrix. The generator matrix G is *systematic* if among its columns can be found the columns of a $k \times k$ identity matrix, in which case G is said to be systematic on those columns or positions. Notice that a standard generator matrix is a special type of systematic generator matrix. If G is a standard generator, then the first k entries of the transmitted codeword $\mathbf{x}G$ contain the message vector $\mathbf{x}$. If G is systematic, then all the entries of the message vector appear among the entries of the transmitted codeword. A subset of the coordinate positions of a linear code is called an *information set* if there is a generator matrix for the code that is systematic on the columns in those positions. We can think of the positions of an information set as carrying the information, while the remaining positions are contributing redundancy. A given code may, however, have many different information sets. A choice of one set is essentially a choice of the corresponding systematic generator for encoding purposes.

standard generator matrix

systematic

information set

EXAMPLES. Consider the generator matrices given in §3.1. The generator matrix for the repetition code is (trivially) standard. For the repetition code, any coordinate position can serve as information set.

The first generator given for the parity check code is standard. Of the two further generators for the $[7, 6]$ parity check code the first is systematic but not standard, and the second is neither. Every set of 6 coordinate positions is an information set.

The generator matrix given for the $[7, 4]$ binary Hamming and $[8, 4]$ extended binary Hamming code are systematic. Indeed in those cases the generator matrix was designed to be systematic on the positions of the information set $\{3, 5, 6, 7\}$. Although it is not clear from our definition, the set of positions $\{1, 2, 3, 4\}$ is indeed an information set for this code,

as the following standard generator matrix for this code indicates:

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

Not every 4-subset of positions is an information set. The definition via check equations guarantees that $\{4, 5, 6, 7\}$ is not an information set, since each codeword has an even number of 1's in these positions. Each particular even weight pattern occurs on these positions in two different codewords, while no odd weight pattern occurs at all.

The first generator given for the $[4, 2]$ ternary Hamming code is standard while the second is systematic but not standard. Again each pair of positions is an information set. (This universality characterizes MDS codes; see Problem 3.2.3.)

It should be noted that, in some references (particularly engineering texts), generator matrices that we have called "standard" are called "systematic" and the more general class that we call "systematic" is not given a specific name.

The rows of a generator matrix form a basis of its row space, the code. Every linear code has a systematic generator matrix, for instance $\mathcal{RREF}(G)$ for any generator G, where the pivot columns are those of an identity matrix. If the code has a standard generator matrix S, then $S = \mathcal{RREF}(G)$. Therefore a code has a standard generator matrix if and only if its generator matrix G has a $\mathcal{RREF}$ in which the pivot columns are the initial columns.

(3.2.1) PROBLEM. *Let C be an $[n, k]$ linear code over F, and let J be a subset of k coordinate positions. For the generator matrix G we write G_J for the $k \times k$ matrix composed of those columns of G indexed by J. Similarly, for the codeword $\mathbf{c}$, we write $\mathbf{c}_J$ for the k-tuple of entries in the positions of J.*

The following are equivalent:

(1) *J is an information set;*

(2) *for each $\mathbf{m} \in F^k$, there is a unique $\mathbf{c} \in C$ with $\mathbf{c}_J = \mathbf{m}$;*

(3) *for every generator matrix G, the matrix G_J is invertible.*

(3.2.2) PROBLEM. *For a nonlinear code C over A, define an information set to be a minimal subset J of the coordinate positions such that no member of $A^{|J|}$ is repeated in these positions. Prove that the dimension of C is a lower bound for $|J|$.*

Two related codes may be different but still share many properties. For instance, if the code D is gotten from C by reversing all codewords (*i.e.*, first entry last, ..., last entry first) then C and D will likely be different but will have many common properties — the same length, minimum distance, dimension, etc. For many purposes we need not distinguish between C and D.

permutation equivalent

Two codes C and D of length n over A are *permutation equivalent* if they are the same up to a uniform permutation of their coordinate entries. (This is often abbreviated to *equivalence*.) That is, there is a permutation σ of the set $\{1, \ldots, n\}$ such that

equivalence

$$(x_1, x_2, \ldots, x_n) \in C \text{ iff } (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}) \in D.$$

Since every linear code has a systematic generator matrix, and a systematic matrix can be changed into a standard matrix by permuting columns, we find that every linear code is equivalent to a code with a standard generator matrix.

Although we do not need the definitions right now, this seems a good time to give three further notions of equivalence for codes defined over fields. Notice that linearity is not required for the various forms of equivalence. In practise regular equivalence is the one of most relevance for codes that are not linear.

DEFINITION. Two codes C and D of length n over the field F are *diagonally equivalent* if they are the same up to the multiplication in each codeword of the i^{th} entry by the nonzero constant α_i, for each i. diagonally equivalent

DEFINITION. Two codes C and D of length n over the field F are *monomially equivalent* if they are the same up to: monomially equivalent
(1) a uniform permutation of their entries (as with regular equivalence);
(2) the multiplication in each codeword of the i^{th} entry by the nonzero constant α_i, for each i.

So monomial equivalence is the union of regular equivalence and diagonal equivalence. For a linear code it corresponds to multiplying column i of a generator matrix by the constant α_i in addition to permuting the columns of the generator.

DEFINITION. Two codes C and D of length n over the field F are *affine equivalent* if they are the same up to: affine equivalent
(1) a uniform permutation of their entries;
(2) the multiplication in each codeword of the i^{th} entry by the nonzero constant α_i, for each i;
(3) translation by a fixed vector of F^n.

Two codes that are affine equivalent have the same size, length, and minimum distance. Here if C is a linear code, then D is a coset of a code monomially equivalent to C.

(3.2.3) PROBLEM. *For $k \neq 0$, prove that the $[n, k]$ linear code C is an MDS code if and only if every subset of k coordinate positions is an information set.*

(3.2.4) PROBLEM. *(THRESHOLD DECODING OF MDS CODES.) Let C be an $[n, k]$ linear MDS code with $k \neq 0$ and generator matrix G. For a set J of coordinate positions, the matrix G_J is that submatrix of G composed of the columns of G that are indexed by the members of J.*

By Problem 3.2.3 every k subset J of coordinate positions is an information set for C, so by Problem 3.2.1 the matrix G_J is always invertible. Indeed if the message k-tuple $\mathbf{m}$ gives rise to the codeword $\mathbf{c} = \mathbf{m}G$, then we can recover $\mathbf{m}$ from $\mathbf{c}$ by $\mathbf{m} = \mathbf{c}_J G_J^{-1}$.

For decoding purposes this means that, for any received vector $\mathbf{r}$, each k subset J of coordinate positions produces a "guess" or "vote" $\widehat{\mathbf{m}}_J = \mathbf{r}_J G_J^{-1}$ as to the identity of the original message $\mathbf{m}$. We choose that k-tuple $\widehat{\mathbf{m}}$ that receives the most votes and then decode $\mathbf{r}$ to the codeword $\widehat{\mathbf{c}} = \widehat{\mathbf{m}}G$.

Suppose that $\mathbf{c} = \mathbf{m}G$ has been transmitted and that $\mathbf{r}$ has been received, e symbol errors having occurred (that is, $d_H(\mathbf{c}, \mathbf{r}) = e$). For k independent variables $x_1, \ldots, x_k$

arranged as a row vector $\mathbf{x} = (x_1, \ldots, x_k)$*, consider the* n *linear equations,* $j = 1, \ldots, n$ *,*

$$\mathsf{Eqn}_j : \qquad \mathbf{r}_j = \mathbf{x}G_j \ ,$$

where $\mathbf{r}_j$ *is the* j*-th entry of received* $\mathbf{r}$ *and* G_j *is the* j*-th column of the matrix* G*.*

(*a*) *Prove that setting* $\mathbf{x}$ *equal to* $\mathbf{m}$ *solves* $n - e$ *of the equations* Eqn_j *. Prove that* $\mathbf{m}$ *gets* $\binom{n-e}{k}$ *votes.*

(*b*) *For any* k*-tuple* $\mathbf{l}$ *that is not equal to* $\mathbf{m}$*, prove that setting* $\mathbf{x}$ *equal to* $\mathbf{l}$ *solves at most* $e + k - 1$ *of the equations* Eqn_j *. Prove that* $\mathbf{l}$ *gets at most* $\binom{e+k-1}{k}$ *votes.*

(*c*) *Prove that, as long as* $2e < n - k + 1$ $(= d_{min}(C))$*, the received vector* $\mathbf{r}$ *will be decoded correctly to* $\mathbf{c}$*.*

(3.2.5) PROBLEM. *Consider the MDS code* C *over the field* $\mathbb{F}_7$ *of integers modulo 7 with generator matrix*

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 6 & 1 & 3 & 5 \end{bmatrix} .$$

Use the method of the previous problem to decode the received vector

$$\mathbf{r} = (1, 3, 6, 5, 4, 2) .$$

3.3 Decoding linear codes

dictionary decoding

A form of decoding always available is *dictionary decoding*. In this we make a list of all possible received words and next to each word write the codeword (or codewords) to which it may be decoded under **MLD**. In decoding, when a word is received we look it up in our "dictionary" and decode it to a codeword listed opposite it. This will almost never be a practical solution.

We now wish to use the structure of linear codes to aid in their decoding. If we are transmitting with a linear code C of length n over F, then we can think of the channel as adding in an *error vector* or *error pattern* (or even *error word*). If $\mathbf{c}$ is transmitted and $\mathbf{x}$ is received, then the channel noise has had the effect of adding to $\mathbf{c}$ the error vector $\mathbf{e} = \mathbf{x} - \mathbf{c} \in F^n$, so that $\mathbf{x} = \mathbf{c} + \mathbf{e}$. The decoding problem is then, given $\mathbf{x}$, estimate at least one of $\mathbf{c}$ and $\mathbf{e}$. The weight of $\mathbf{e}$ is the number of positions in which $\mathbf{c}$ and $\mathbf{x}$ differ; so, when using an $mSC(p)$ (with $p < 1/m$) and decoding under **MLD**, we are looking for an error pattern $\mathbf{e}$ of minimum weight.

error vector
error pattern
error word

From the definition of $\mathbf{e}$, we learn that the received vector $\mathbf{x}$ and the error pattern $\mathbf{e}$ belong to the same coset $\mathbf{x} + C = \mathbf{e} + C$. While we do not know $\mathbf{x}$ ahead of time, the cosets of C can be calculated in advance. We look for vectors of minimum weight in each coset. Such a vector is caller a *coset leader*. Notice that while the minimum weight of a coset is well-defined, there may be more than one vector of that weight in the coset, that is, there may be more than one coset leader. Usually we choose and fix one of the coset leaders. Always $\mathbf{0}$ is the unique coset leader for the code itself.

coset leader

We first describe the general technique of decoding with coset leaders and then give two methods for its implementation. When the word $\mathbf{x}$ is received, we do not know the actual error that was introduced; but we do know that it

belongs to the coset $\mathbf{x} + C$. Thus if $\widehat{\mathbf{e}}$ is the coset leader chosen for this coset, then $\widehat{\mathbf{e}}$ is one of the most likely error patterns; and we guess that it was the actual error. (In fact $\widehat{\mathbf{e}}$ is the unique most likely error pattern if it is the unique leader of its coset.) We decode $\mathbf{x}$ to the codeword $\widehat{\mathbf{c}} = \mathbf{x} - \widehat{\mathbf{e}}$. With coset leader decoding, the error patterns that are corrected are exactly those that are the chosen coset leaders. In particular, the code will be e-error-correcting if and only if every vector of weight at most e is the unique leader of its coset.

Coset leader decoding is an **MDD** algorithm for linear codes over fields of size m. Therefore knowledge of the coset leaders for C makes it easy to calculate $\mathcal{P}_C$ on an $mSC(p)$. Indeed, an error pattern will be corrected if and only if it is a chosen coset leader. Thus, for $mSC(p)$ with $q = 1 - (m-1)p > p$, we have

$$\mathcal{P}_C = \mathcal{P}_C(\mathbf{MDD}) = 1 - \left(\sum_{i=0}^{n} a_i p^i q^{n-i}\right),$$

where a_i is the number of cosets of C with coset leader of weight i.

(3.3.1) PROBLEM. *If $\mathbf{x}$ and $\mathbf{y}$ are binary vectors of length n, then we write $\mathbf{x} \preceq \mathbf{y}$ to indicate that $\mathbf{x}$ has a 0 in every position that $\mathbf{y}$ has a 0 (but $\mathbf{x}$ may have 0's in places that $\mathbf{y}$ has 1's.) Equivalently, everywhere $\mathbf{x}$ has a 1, $\mathbf{y}$ also has a 1, but $\mathbf{y}$ may have more 1's. For instance*

$$(0,0,0,1,1,0,1,0) \preceq (0,1,0,1,1,0,1,1)\,.$$

(a) Let $\mathbf{x}$ and $\mathbf{y}$ be binary n-tuples, and set $\mathbf{f} = \mathbf{x} + \mathbf{y}$. Prove that $\mathbf{x} \preceq \mathbf{y}$ if and only if $w_H(\mathbf{y}) = w_H(\mathbf{x}) + w_H(\mathbf{f})$.

(b) Let C be a binary linear code of length n. Prove that if $\mathbf{y}$ is a coset leader for the coset $\mathbf{y} + C$ and $\mathbf{x} \preceq \mathbf{y}$, then $\mathbf{x}$ is also a coset leader for the coset $\mathbf{x} + C$.

standard array decoding

Our first method for coset leader decoding is *standard array decoding*. Set $K = |C|$, the cardinality of C, and $R = |F^n|/|C|$, the number of distinct cosets of C. Enumerate the codewords:

$$C = \{\ \mathbf{c}_1 = \mathbf{0},\ \mathbf{c}_2,\ \ldots\ , \mathbf{c}_K\}$$

and coset leaders:

$$\{\ \mathbf{e}_1 = \mathbf{0},\ \mathbf{e}_2,\ \ldots\ , \mathbf{e}_R\},$$

one coset leader for each coset of C in F^n. We form a large array, the *standard array*, whose first row contains the codewords, and first column contains the coset leaders, and in general has $\mathbf{c}_j + \mathbf{e}_i$ as its i, j entry. The i^{th} row of the standard array is the coset $\mathbf{e}_i + C$. Thus every n-tuple of F^n is contained exactly once in the array.

To decode using the standard array, when $\mathbf{x}$ is received, look it up in the array. If it is in the i, j position, then we have $\mathbf{x} = \mathbf{c}_j + \mathbf{e}_i$. In this case we assume that the introduced error vector was $\mathbf{e}_i$ and decode $\mathbf{x}$ to $\widehat{\mathbf{c}} = \mathbf{c}_j$.

Standard array decoding is not of much practical value as it involves storage of the large array as well as random access searches through the array. It does

have historical and theoretical value, because it illustrates the important general fact that code structure can be exploited to design decoding algorithms that are more efficient than dictionary decoding.

syndrome decoding

The second method of coset leader decoding is *syndrome decoding*, where the dual code and check matrices are used. Let H be a check matrix for the $[n, k]$ linear code C. We have already mentioned that the vector $\mathbf{x}$ is in the code C if and only if the matrix product $H\mathbf{x}^\top$ equals $\mathbf{0}$. For any received vector $\mathbf{x}$, the length $r = n - k$ column vector $H\mathbf{x}^\top$ is a measure of whether on not the n-tuple $\mathbf{x}$ belongs to the code. The column r-tuple $H\mathbf{x}^\top$ is the *syndrome* of the n-tuple $\mathbf{x}$. According to the "Pocket Oxford Dictionary," a syndrome is generally a "characteristic combination of opinions." The syndrome voices information about the error vector. Syndrome decoding is error oriented, using the opinions voiced by the syndrome vector to identify the appropriate error vector.

syndrome

As the syndrome of a codeword is $\mathbf{0}$, two vectors $\mathbf{x}$ and $\mathbf{e}$ that differ by a codeword $\mathbf{c}$ will have the same syndrome:

$$H\mathbf{x}^\top = H(\mathbf{c} + \mathbf{e})^\top = \mathbf{0} + H\mathbf{e}^\top = H\mathbf{e}^\top$$

That is, syndromes are constant on cosets of C in F^n. Equally well, distinct cosets have different syndromes since the difference of vectors from distinct cosets is not a codeword and so has nonzero syndrome.

We interpret the above equation as saying that a received vector $\mathbf{x}$ and the corresponding error vector $\mathbf{e}$ introduced by the channel will have the same syndrome, namely that of the coset to which they both belong. Instead of storing the entire standard array, we need only store a *syndrome dictionary* (or syndrome table) containing all possible syndromes $\{\mathbf{s}_1 = \mathbf{0}, \ \ldots \ , \ \mathbf{s}_R\}$ together with coset leaders $\{\mathbf{e}_1 = \mathbf{0}, \ \ldots \ , \ \mathbf{e}_R\}$ such that $H\mathbf{e}_i^\top = \mathbf{s}_i$. In decoding, when $\mathbf{x}$ is received, first calculate the syndrome $\mathbf{s} = H\mathbf{x}^\top$. Next look up $\mathbf{s}$ in the syndrome dictionary as $\mathbf{s} = \mathbf{s}_i$. Finally decode $\mathbf{x}$ to $\widehat{\mathbf{c}} = \mathbf{x} - \mathbf{e}_i$.

syndrome dictionary

EXAMPLE. Consider the $[4, 2]$ ternary Hamming code with check matrix

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

The syndromes are therefore column vectors of length 2. For instance, the received vector $\mathbf{x} = (1, 2, 1, 1)$ has syndrome

$$H\mathbf{x}^\top = \begin{pmatrix} 1+2+2+0 \\ 0+2+1+1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} .$$

To decode using syndromes we first write out our syndrome dictionary,

the first column containing the transposes of all possible syndromes.

syndrome transpose	coset leader
00	0000
01	0001
02	0002
10	1000
11	0100
12	0002
20	2000
21	0010
22	0200

It is not necessary to list out all cosets of the code to make this dictionary. Instead notice that two words of $\mathbb{F}_3^4$ are in the same coset if and only if their difference is a codeword. So, for instance, not only must $(0,0,0,1)$, $(0,0,0,2)$, and $(0,2,0,0)$ all be of minimum weight in their respective cosets; but they belong to different cosets. (Subtracting one of them from another gives a word of weight less than 3, not a codeword since the minimum weight of the Hamming code is 3.) The transposed syndromes are then calculated as, respectively, $(0,1)$, $(0,2)$, and $(2,2)$; and the results are recorded in the dictionary.

To decode our received vector $\mathbf{x} = (1,2,1,1)$ we first calculate, as before, its transposed syndrome $(2,1)$. We then look up this syndrome in our dictionary and discover that the corresponding coset leader is $\widehat{\mathbf{e}} = (0,0,1,0)$. We therefore assume that this is the error that occurred and decode $\mathbf{x}$ to the codeword

$$\widehat{\mathbf{c}} = \mathbf{x} - \widehat{\mathbf{e}} = (1,2,1,1) - (0,0,1,0) = (1,2,0,1) \ .$$

It may sometimes be more convenient to define syndromes and do syndrome decoding relative to a control matrix H rather than a check matrix.

Syndrome decoding does not suffer from many of the failings of standard array decoding. The syndrome dictionary is much smaller than the standard array for storage purposes; and it can be ordered lexicographically, so that searches can be done linearly. Still syndrome decoding in this dictionary form is too general to be of much practical use. Certain practical decoding algorithms do employ partial syndrome dictionaries that list only the most common syndromes. Syndrome decoding is also the paradigm for many genuine decoding techniques. To each received vector we associate some kind of "syndrome." The properties of the specific code then are used in the passage from syndrome to error vector and decoded word. The decoding method for the $[7,4]$ Hamming code as given by Shannon in Example 1.3.3 is a type of syndrome decoding, since he has arranged the columns of the check matrix H (given on page 37) to contain the binary numbers in order. The calculated syndrome $\alpha\beta\gamma$ is therefore associated with the coset whose leader has a 1 in the $\alpha\beta\gamma^{\text{th}}$ position (read in binary) and 0 elsewhere. We decode assuming an error in this position.

(3.3.2) PROBLEM. *(a) Give a syndrome table for the* $[8,4]$ *extended binary Hamming code* E *with the following check matrix (and generator matrix—the code is self dual):*

$$EL_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

(b) Use your table to decode the received word:

$$(0,0,1,0,0,1,1,0)\,.$$

(c) Use your table to decode the received word:

$$(0,1,1,1,1,1,0,1)\,.$$

We now consider using syndrome decoding and check matrices to correct erasures rather than errors. (See Problem 2.2.4.) Remember that erasures occur as a consequence of soft quantization of received symbols. We allow transmitted symbols from the alphabet A to be received as members of A or as ?, the erasure symbol. Alternatively we may think of erasures as symbol errors whose locations are known. Under this second interpretation, we might receive a word from the alphabet A but with certain positions flagged as being unreliable. These flagged positions are then the erasure locations. The two views of erasures are equivalent. Indeed each occurrence of ? may filled arbitrarily by an alphabet letter (typically 0 for a linear code) and then flagged as unreliable. Conversely each flagged symbol can be replaced by ?, the erasure symbol. Which point of view is the best will depend upon the particular situation.

Since C contains codewords of weight $d = d_{min}(C)$ as well as $\mathbf{0}$ of weight 0, we could never hope to correct d erasures; but we can decode up to $d-1$ erasures correctly.

(3.3.3) PROPOSITION. *Let* C *be an* $[n,k,d]$ *linear code over* F *with check matrix* H *whose rows are* $\mathbf{h}_i$*, for* $i = 1,\ldots,r = n-k$*. Let* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *be an* n*-tuple of indeterminates.*

Assume the codeword $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *is transmitted, and we receive the vector* $\mathbf{p} = (p_1, p_2, \ldots, p_n) \in F^n$ *with the entries* p_l*, for* $l \in L$*, flagged as erasures but* $p_j = c_j$*, for* $j \notin L$*.*

If $|L| \leq d-1$*, then the set of equations in the unknowns* x_i

$$\begin{aligned} \mathbf{h}_i \cdot \mathbf{x} &= \mathbf{h}_i \cdot \mathbf{p} \text{ for } i = 1,\ldots,r \\ x_j &= 0 \text{ for } j \notin L \end{aligned} \qquad (*)$$

has as its unique solution the erasure vector

$$\mathbf{x} = \mathbf{c} - \mathbf{p} = \mathbf{e}\,.$$

Therefore by solving the equations $(*)$ *we can decode all patterns of up to* $d-1$ *erasures in codewords of* C*.*

PROOF. This set of equations has at least one solution, namely the actual erasure vector $\mathbf{e} = \mathbf{c} - \mathbf{p}$. If $\mathbf{e}'$ is any solution of the equations $(*)$ then $\mathbf{c}' = \mathbf{e} - \mathbf{e}'$ has syndrome $\mathbf{0}$ and equals 0 off L. Therefore $\mathbf{c}'$ is a codeword of weight at most $d - 1$ and so must be $\mathbf{0}$. We conclude that $\mathbf{e} = \mathbf{e}'$, and the set of equations $(*)$ has the unique solution $\mathbf{x} = \mathbf{e}$. □

The equations $(*)$ give $n + r - |L|$ linear equations in the n unknowns, where $r \geq d - 1 \geq |L|$ (by the Singleton bound 3.1.14). By the proposition, the solution is unique; so the system has rank n. The last $n - |L|$ syndrome equations of $(*)$ are clearly linearly independent; so we may delete some of the first r equations to reduce $(*)$ to a system of n equations in n unknowns with a unique solution. These equations can be solved by Gaussian elimination; so the number of operations required is at worst on the order of n^3, a respectably small number. Indeed the set of equations is essentially triangular, so the complexity is actually on the order of n^2.

The algorithm of the proposition is an effective method for correcting erasures in any linear code. This can in turn be helpful when decoding errors as well as erasures. We may concentrate our efforts upon designing an algorithm that locates errors. After the errors have been located, they can be thought of as flagged erasures and their values found with the algorithm of Proposition 3.3.3. Complexity on the order of n^2 is not a bottleneck in any of the presently popular algorithms.

Chapter 4

Hamming Codes

In the late 1940's Claude Shannon was developing information theory and coding as a mathematical model for communication. At the same time, Richard Hamming, a colleague of Shannon's at Bell Laboratories, found a need for error correction in his work on computers. Parity checking was already being used to detect errors in the calculations of the relay-based computers of the day, and Hamming realized that a more sophisticated pattern of parity checking allowed the correction of single errors along with the detection of double errors. The codes that Hamming devised, the single-error-correcting binary Hamming codes and their single-error-correcting, double-error-detecting extended versions marked the beginning of coding theory. These codes remain important to this day, for theoretical and practical reasons as well as historical.

4.1 Basics

Denote by L_3 the check matrix that we have been using to describe the $[7, 4]$ Hamming code:

$$L_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

It has among its columns each nonzero triple from $\mathbb{F}_2^3$ exactly once. From this and Lemma 3.1.12, we were able to prove that the $[7, 4]$ Hamming code has minimum distance 3. This suggests a general method for building binary Hamming codes. For any r, construct a binary $r \times 2^r - 1$ matrix H such that each nonzero binary r-tuple occurs exactly once as a column of H. Any code with such a check matrix H is a *binary Hamming code* of redundancy r, denoted $Ham_r(2)$. Thus the $[7, 4]$ code is a Hamming code $Ham_3(2)$. Each binary Hamming code has minimum weight and distance 3, since as before there are no columns $\mathbf{0}$ and no pair of identical columns. That is, no pair of columns is linearly dependent, while any two columns sum to a third column, giving a triple of linearly dependent columns. Lemma 3.1.12 again applies.

binary Hamming code

As defined, any code that is equivalent to a binary Hamming code is itself a Hamming code, since any permutation of coordinate positions corresponds to a permutation of the columns of the associated check matrix. The new check matrix is still a census of the nonzero r-tuples. Different codes and check matrices may be selected to suit different purposes.

EXAMPLES. The following are check matrices for two $[15, 11]$ binary Hamming codes $Ham_4(2)$:

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

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

The first is the check matrix for a code which has a generator matrix in standard form (see page 35 and Problem 4.1.9 below). The second matrix checks a code which has no generator in standard form, since, for instance, (000000000001111) is a codeword.

The second of the two example check matrices, which we will denote L_4, is the counterpart of the matrix L_3 above, in that its column i contains the binary representation of i. For each positive integer r, let L_r be the $r \times 2^r - 1$ matrix whose i^{th} column is the binary representation of the integer i (with least significant digit at the bottom). Then L_r is the check matrix for a Hamming code $Ham_r(2)$. We call L_r a *lexicographic check matrix*.

lexicographic check matrix

EXAMPLES. We have given L_3 and L_4 above. We also have the smaller cases

$$L_1 = [1] \quad \text{and} \quad L_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

which are check matrices for, respectively, the degenerate Hamming code $Ham_1(2) = \{\mathbf{0}\}$ and $Ham_2(2)$, the repetition code of length 3.

For a binary Hamming code with lexicographic check matrix L_r, we have an easy version of syndrome decoding available, similar to that for $Ham_3(2)$ discussed earlier and presented by Shannon under Example 1.3.3. If the vector $\mathbf{x}$ has been received, then to decode we first calculate the syndrome $\mathbf{s} = L_r\mathbf{x}^\top$. If $\mathbf{s}$ is $\mathbf{0}$, then $\mathbf{x}$ is a codeword and no further decoding is required. If $\mathbf{s}$ is not $\mathbf{0}$, then it is the binary representation of some integer, j say, between 1 and $2^r - 1$. We decode by assuming that a single error has occurred in position j of $\mathbf{x}$.

If we add an overall parity check bit to a binary Hamming code $Ham_r(2)$, then the minimum distance is increased to 4. We then have an *extended Hamming code*, denoted $EHam_r(2)$. By Problem 2.2.3 this is a 1-error-correcting, 2-error-detecting binary linear $[2^r, 2^r - r]$ code, as originally constructed by Hamming.

extended Hamming code

Begin with the Hamming code $Ham_r(2)$ given by the lexicographic check matrix L_r and extend by adding an overall parity check bit at the front of each

codeword. A check matrix EL_r for this extended Hamming code $EHam_r(2)$ can be gotten by adding a column r-tuple $\mathbf{0}$ at the beginning of L_r and then adding at the bottom the vector $\mathbf{1}$ composed entirely of 1's. Here we do not follow the usual convention, that of Example 1.3.4, where the parity check bit is added at the end. Instead it is more natural to add the column $\mathbf{0}$, the binary representation of the integer 0, at the front of the lexicographic matrix.

EXAMPLES.

$$EL_1 = \left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & 1 \end{array}\right] \quad \text{and} \quad EL_2 = \left[\begin{array}{c|ccc} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 \end{array}\right]$$

$$EL_3 = \left[\begin{array}{c|ccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

$$EL_4 = \left[\begin{array}{c|cccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

To see that EL_r is a check matrix for the extended lexicographic Hamming code, first note that its $r+1$ rows are linearly independent rows; so at least it has the right rank. If $\mathbf{c} = (c_1, \ldots, c_n)$ is a word from the Hamming code, then the corresponding extended word is $\mathbf{c}' = (c_0, c_1, \ldots, c_n)$ where $c_0 = \sum_{i=1}^{n} c_i$ is the overall parity check symbol. The codeword $\mathbf{c}'$ has dot product 0 with each of the first r rows, since its original form $\mathbf{c}$ in the Hamming code has dot product 0 with the corresponding row of L_r. Also the dot product of $\mathbf{c}'$ with $\mathbf{1}$ is $c_0 + c_1 + \cdots + c_n = (\sum_{i=1}^{n} c_i) + c_1 + \cdots + c_n = 0$. Therefore EL_r is indeed a check matrix for the extended Hamming code as described.

We can think of our construction of binary Hamming codes as a greedy construction. We begin with an integer r and try to construct the check matrix for a 1-error-correcting binary linear code of redundancy r that, subject to this, is of maximal length. We add new columns to a check matrix, being careful at each stage not to reduce the minimum distance below three. By Lemma 3.1.12 we can do this provided at each stage we add a new column that is not linearly dependent upon any previous column. As our field is $\mathbb{F}_2$, this amounts to avoiding repeat columns and the $\mathbf{0}$ column.

This approach to Hamming codes easily extends to linear codes over finite fields $\mathbb{F}_q$ other than $\mathbb{F}_2$. Again, begin with r and construct a check matrix for a long 1-error-correcting linear code over $\mathbb{F}_q$ with redundancy r. Each time we add a new column to our matrix, we must take care that it does not depend linearly upon any previously chosen column. We avoid $\mathbf{0}$ always, so initially we can choose from among $q^r - 1$ nonzero column r-tuples. When we add in a nonzero column, we remove not just it from further consideration but each of its $q-1$ multiples by the nonzero elements of $\mathbb{F}_q$. Therefore the maximum length possible is $(q^r - 1)/(q-1)$. One easy way of constructing such a matrix of this

maximum length is to choose as columns all nonzero r-tuples whose top-most nonzero entry is 1. A linear code over the finite field $\mathbb{F}_q$ is a *Hamming code* of redundancy r, written $Ham_r(q)$, if it has a check matrix whose collection of columns contains a unique nonzero scalar multiple of each nonzero r-tuple from $\mathbb{F}_q$. In particular any code that is monomially equivalent to a Hamming code is again a Hamming code. Note that this general definition includes that for binary Hamming codes given above.

Hamming code

EXAMPLES.

(i) Our original description of $Ham_2(3)$, a ternary Hamming code of redundancy 2, was in terms of the check matrix

$$\begin{bmatrix} 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

A ternary Hamming code of redundancy 2 can also be constructed from the "lexicographic" check matrix

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}.$$

(ii) With $q = 9$, $r = 2$, and $n = (9^2 - 1)/(9 - 1) = 10$, one check matrix for a Hamming code $Ham_2(9)$ of length 10 is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & i & 1+i & 2+i & 2i & 1+2i & 2+2i \end{bmatrix},$$

where i is a square root of $2 = -1$ in $\mathbb{F}_9$.

(4.1.1) THEOREM. *A Hamming code of redundancy $r(\geq 2)$ over the field F, $|F| = q$, is a linear*

$$\left[\frac{q^r - 1}{q - 1}, \frac{q^r - 1}{q - 1} - r, 3\right]$$

code and is a perfect 1*-error-correcting code.*

PROOF. Set $n = (q^r - 1)/(q - 1)$, the length of the Hamming code C. As the code has redundancy r, its dimension is $n - r$. As discussed above, the argument applying Lemma 3.1.12 to prove that $d_{min}(C) = 3$ for binary codes goes over to the general case; so C corrects single errors. The Sphere Packing Condition 2.2.6 for correcting $e = 1$ error then says

$$|C| \cdot |S_1(*)| \leq |A^n|.$$

Here this takes the form

$$q^{n-r} \cdot \big(1 + (q-1)n\big) \leq q^n.$$

That is,

$$q^{n-r} \cdot \big(1 + (q-1)n\big) = q^{n-r} \cdot \big(1 + (q-1)\frac{q^r - 1}{q-1}\big) = q^{n-r} \cdot q^r \leq q^n.$$

Thus we in fact have equality in the Sphere Packing Condition 2.2.6 and the Sphere Packing Bound 2.3.6. □

(4.1.2) PROBLEM. *Prove that the family of all q-ary Hamming codes is asymptotically bad, being associated with the point $(0,1)$ of the q-ary code region.*

(4.1.3) PROBLEM. *Prove that a linear*

$$\left[\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1}-r, 3\right]$$

code over the field $\mathbb{F}_q$ (a field with q elements) is a Hamming code.

Problem 4.1.3 says that, among linear codes, the Hamming codes are characterized by their parameters — length, dimension, and distance. This is not the case if we drop the assumption of linearity. Of course any coset of a Hamming code $Ham_r(q)$ has the same parameters as the code. For $q = 2$ and for $r \leq 3$, the converse is true. This is trivial for $r = 1, 2$ and the case $r = 3$ is given as the next problem. For $q = 2$ and $r \geq 4$ nonlinear codes with Hamming parameters exist that are not a coset of any Hamming code. Examples are constructed in Problem 4.1.7 below. For $q > 2$ and $r \geq 2$, nonlinear examples exist as well and can be constructed in a manner related to that of Problem 4.1.7.

(4.1.4) PROBLEM. *Prove that a 1-error-correcting binary code of length 7 that contains 16 vectors must be a coset of a $[7,4]$ Hamming code.*

(4.1.5) PROBLEM. *Let C_1 be an $[n, k_1, d_1]$ linear code over F, and let C_2 be a $[n, k_2, d_2]$ linear code code over F. Form the code*

$$C = \{\ (\mathbf{y}\,;\mathbf{x}+\mathbf{y}) \mid \mathbf{x} \in C_1,\ \mathbf{y} \in C_2\ \}.$$

(a) If C_1 has generator matrix G_1 and C_2 has generator matrix G_2, prove that C is a $[2n, k_1 + k_2]$ linear code over F with generator matrix

$$G = \begin{bmatrix} \mathbf{0} & G_1 \\ G_2 & G_2 \end{bmatrix},$$

where the upper left $\mathbf{0}$ is a $k_1 \times n$ matrix entirely composed of 0's.

(b) Prove that $d_{min}(C) = \min(d_1, 2d_2)$.

(c) Under the additional assumption $d_1 > 2d_2$, prove that all codewords of minimum weight in C have the form $(\mathbf{y}\,;\mathbf{y})$, where $\mathbf{y}$ has minimum weight in C_2.

(4.1.6) PROBLEM. *Formulate and prove the appropriate version of Problem 4.1.5 for nonlinear C_1 and C_2.*

(4.1.7) PROBLEM. *In Problem 4.1.5 let $C_1 = Ham_{r-1}(2)$ and $C_2 = \mathbb{F}_2^n$ where $n = 2^{r-1}-1$. Then C has length $2n = 2^r-2$, dimension $2^{r-1}-1-(r-1)+2^{r-1} = 2^r-r$, and minimum distance 2. Furthermore all codewords of weight 2 in C have the shape $(\mathbf{y}\,;\mathbf{y})$, for $\mathbf{y} \in \mathbb{F}_2^n$ of weight 1. Consider now the code*

$$C^* = \{\ (\mathbf{y}\,;\mathbf{x}+\mathbf{y}\,;c) \mid \mathbf{x} \in Ham_{r-1}(2),\ \mathbf{y} \in \mathbb{F}_2^n,\ c = \mathbf{y}\cdot\mathbf{1} + f(\mathbf{x})\ \},$$

where $\mathbf{1} \in \mathbb{F}_2^n$ is the vector with all coordinates equal to 1, $\mathbf{y}\cdot\mathbf{1}$ is the usual dot product, and f is any function from $Ham_{r-1}(2)$ to $\mathbb{F}_2$.

(*a*) *Prove that* C^* *is a code with the same length, size, and minimum distance as* $Ham_r(2)$.

(*b*) *Prove that* C^* *is linear if and only if the function* f *is a linear map on* $Ham_{r-1}(2)$ *(that is, for all* $\mathbf{x_1}, \mathbf{x_2} \in Ham_{r-1}(2)$*, we have* $f(\mathbf{x_1})+f(\mathbf{x_2}) = f(\mathbf{x_1}+\mathbf{x_2})$*.)*

(*c*) *Prove that, for all* $r \geq 4$*, there exist binary codes* C^* *with the same parameters as the Hamming codes* $Ham_r(2)$ *that are not equal to a coset of any Hamming code.*

Any check matrix H for a Hamming code $Ham_r(q)$ can be easily used for syndrome decoding. Upon encountering a nonzero syndrome $\mathbf{s} = H\mathbf{x}^\top$, we must survey the columns of H for a scalar multiple of $\mathbf{s}$. If $\mathbf{s}$ is $\alpha\mathbf{h}$, where $\mathbf{h}$ is the i^{th} column of H, then we assume that the error vector had weight one with entry α in its i^{th} coordinate, and decode accordingly.

(4.1.8) PROBLEM. *Consider the ternary* $[13, 10]$ *Hamming code with check matrix*

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{bmatrix} .$$

Decode the received word

$$(2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1) .$$

(4.1.9) PROBLEM. *Consider a binary* $[n, k]$ *linear code* C_r *with redundancy* $r = n-k$ *and standard generator matrix* $\left[I_{k\times k} \mid A \right]$*, for some* $k \times r$ *binary matrix* A.

(*a*) *Prove that* $d_{min}(C) \geq 3$ *if and only if the rows of* A *are pairwise distinct* r*-tuples of weight at least* 2.

(*b*) *Using the result from* (*a*)*, prove that the maximum possible number of rows in* A *for a* 1*-error-correcting code* C_r *is* $2^r - 1 - r$ *and that, in this case,* C *is a Hamming code* $Ham_r(2)$.

In constructing codes with a fixed minimum distance d we hope for codes whose dimension is a large fraction of their length or, equivalently, whose redundancy is small compared to their length. Let $s_q(n, f) = \sum_{i=0}^{f} \binom{n}{i}(q-1)^i$ be the volume of a sphere of radius f in $\mathbb{F}_q^n$. The Gilbert-Varshamov Theorem 2.2.7 proved, by a greedy construction of codewords, that there is in $\mathbb{F}_q^n$ a code C with minimum distance d and satisfying $|C| \geq q^n/s_q(n, d-1)$. That is, we can find a code C with distance d and length n whose redundancy $r = n - \log_q(|C|)$ is bounded above by

$$q^r \leq s_q(n, d-1) .$$

In this section we have used greedy constructions instead to construct the check matrices of Hamming codes. We pursue this in Problem 4.1.10 and find, via a greedy construction of a check matrix, that there is a linear code C with minimum distance d and redundancy r whose length n is bounded below by

$$q^r \leq s_q(n, d-2) .$$

This small improvement may be relevant in specific cases; but, since in the limit $(d-1)/n$ and $(d-2)/n$ are indistinguishable, the asymptotic bound of Theorem

2.3.7 is not affected. Although the bounds have a very similar appearance, the two constructions are essentially different in character. The first endeavors to constructs a code of small redundancy with given length, while the second tries to construct a long code with given redundancy.

(4.1.10) PROBLEM. *(a) Let H be an $r \times m$ matrix with entries from $\mathbb{F}_q$. For a fixed positive integer d, prove that the number of column r-tuples that linearly depend upon some subset of $d-2$ columns from H is at most $\sum_{i=0}^{d-2} \binom{m}{i}(q-1)^i = s_q(m, d-2)$.*

(b) (LINEAR GILBERT-VARSHAMOV BOUND) *Prove that there exists a linear code C over $\mathbb{F}_q$ of minimum distance d and redundancy r whose length n satisfies*

$$q^r \leq s_q(n, d-2) \ .$$

4.2 Hamming codes and data compression

Hamming codes can also be used for data compression allowing a small amount of distortion (loss of information) by "running the machine backwards."

Choose a generator matrix G for a Hamming code $Ham_r(q)$ of length n over F. Each n-tuple $\mathbf{x}$ from F^n is at distance at most 1 from a unique codeword $\mathbf{c}$ of $Ham_r(q)$. Instead of storing the n-tuple $\mathbf{x}$, store the smaller message $(n-r)$-tuple $\mathbf{m}$, the unique solution to $\mathbf{m}G = \mathbf{c}$. At decompression, the stored message $\mathbf{m}$ is "encoded" to $\mathbf{c} = \mathbf{m}G$, which differs from the original data vector $\mathbf{x}$ in at most one position. This works because spheres of radius one around the codewords of $Ham_r(q)$ cover the whole codespace F^n.

Consider a code C of length n over the alphabet A. In general the *covering radius* of a code C, denoted $cr(C)$, is the smallest number r such that the spheres of radius r around the codewords of C cover the entire codespace A^n, that is, $A^n = \bigcup_{\mathbf{c} \in C} S_r(\mathbf{c})$. Data compression problems are basically dual to those of error correction. A good code for correcting errors has a large number of words but still has large minimum distance. A good data compression code has a small number of words but still has a small covering radius. As with correction, these are conflicting goals.

covering radius

Data compression questions have been considered for almost as long as those of error correction. Sometimes they have been phrased in terms of the "football pools" or lottery problem. Typically in the lottery, each ticket purchased contains a collection of numbers (selected by the purchaser and from some fixed range A). Whether or not a given ticket is a winner or loser depends upon how well its numbers match those of a master ticket which is selected at a later time. If in order to win it is not necessary for a given ticket to match all of the master lottery numbers but only miss at most f of them, then a question arises. What is the smallest number of lottery tickets I must choose and buy in order to guarantee that I have a ticket on which at most f numbers are wrong? What is being sought is a small code (the collection of purchased lottery tickets) that has covering radius at most f. For the football pools problem, the alphabet A is ternary, since each match result being predicted has three possible outcomes: 'win' (for the home team, say), 'lose', or 'draw'.

For a Hamming code, the covering radius is 1. Indeed, for any perfect e-error-correcting code, the covering radius is e.

(4.2.1) PROPOSITION. *Let C be an e-error-correcting code. Then $cr(C) \geq e$ with equality if and only if C is a perfect e-error-correcting code.*

PROOF. As C is an e-error-correcting code, the spheres of radius e around codewords are pairwise disjoint. Therefore the spheres of radius $e-1$ around codewords do not cover the whole space. Thus $cr(C) > e-1$, whence $cr(C) \geq e$. If we have equality, then we must have equality in the Sphere Packing Bound 2.2.6, hence the code is perfect. □

(4.2.2) PROPOSITION. *The covering radius of the linear code C is equal to the maximum weight of a coset leader.*

PROOF. The coset of the word $-\mathbf{x}$ consists of the sum of $-\mathbf{x}$ with each individual codeword of C; so the weights of the coset members give the distances of $\mathbf{x}$ from the various codewords. The minimal such weight is thus the distance of $\mathbf{x}$ from the code and also the weight of a coset leader. The maximum weight of a coset leader is therefore the largest distance of any word $\mathbf{x}$ from the code. □

As with d_{min}, the covering radius of a code is, in general, difficult to compute. The following problem, reminiscent of Problem 4.1.5, can be of great help.

(4.2.3) PROBLEM. *Let the $[n_1, k_1]$ linear code C_1 over F have generator matrix G_1, and let the $[n_2, k_2]$ linear code C_2 over F have generator matrix G_2. Consider the $[n_1 + n_2, k_1 + k_2]$ linear code C over F with generator matrix*

$$G = \begin{bmatrix} \mathbf{0} & G_1 \\ G_2 & * \end{bmatrix},$$

where the upper left $\mathbf{0}$ is a $k_1 \times n_2$ matrix of 0's and the lower right $*$ is an arbitrary $k_2 \times n_1$ matrix with entries from F.

Prove that $cr(C) \leq cr(C_1) + cr(C_2)$.

4.3 First order Reed-Muller codes

In 1954, I.S. Reed and D.E. Muller introduced independently a class of binary codes of length 2^m, for any integer m, associated with Boolean logic. The first of these codes, for each m, fits naturally into the context of Hamming codes.

first order Reed-Muller code

A code dual to a binary extended Hamming code is called a *first order Reed-Muller code*, denoted $RM(1, m)$ where m is the redundancy of the associated Hamming code. Any code that is equivalent to a first order Reed-Muller code is also first order Reed-Muller. We shall concentrate on the specific code $RM(1, m)$ with generator matrix EL_m.

shortened first order Reed-Muller code
simplex code

The associated dual Hamming code is sometimes called a *shortened first order Reed-Muller code* or a *simplex code*. The dual Hamming code can be

easily recovered from $RM(1,m)$. Indeed by first choosing all the codewords of $RM(1,m)$ that have a 0 in their first coordinate position and then deleting this now useless coordinate, we find the dual Hamming code. This is clear when we consider how the matrix EL_m was constructed by bordering the matrix L_m, the generator matrix of the dual lexicographic Hamming code. (See page 51.)

Having earlier constructed the generator EL_m as a matrix in bordered block form, we now examine it again, but blocked in a different manner. Notice that $RM(1,1) = \mathbb{F}_2^2$, $RM(1,2)$ is the parity check code of length 4, and $RM(1,3)$ is a self-dual extended $[8,4]$ Hamming code.

EXAMPLES.

$$EL_1 = \left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & 1 \end{array}\right] \quad \text{and} \quad EL_2 = \left[\begin{array}{cc|cc} 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right]$$

$$EL_3 = \left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

$$EL_4 = \left[\begin{array}{cccccccc|cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

For i between 2^{m-1} and 2^m, the column m-tuple containing the binary representation of i is just that for $i - 2^{m-1}$ with its leading 0 replaced by a 1. Therefore, if we ignore the top row of EL_m, then the remaining m rows consist of an $m \times 2^{m-1}$ matrix repeated twice. Indeed this repeated matrix is nothing other than EL_{m-1}. We now observe the recursive construction:

$$EL_m = \left[\begin{array}{c|c} 0 \cdots 0 & 1 \cdots 1 \\ \hline EL_{m-1} & EL_{m-1} \end{array}\right] .$$

(4.3.1) THEOREM. *For each m, the first order Reed-Muller code $RM(1,m)$ is a binary linear $[2^m, m+1, 2^{m-1}]$ code.*

PROOF. Certainly $RM(1,m)$ is linear of length 2^m, and its dimension $m+1$ is evident from the generator matrix EL_m. From their generator matrices EL_1 and EL_2, it is easy to see that $RM(1,1)$ $(= \mathbb{F}_2^2)$ and $RM(1,2)$ (the parity check code of length 4) both have minimum distance 2^{m-1}.

We now verify the minimum distance in general by induction, assuming that we already have $d_{min}(RM(1,m-1)) = 2^{m-2}$. Let C_1 be $RM(1,m-1)$ with minimum distance $d_1 = 2^{m-2}$, and let C_2 be the repetition code of length 2^{m-1}, whose minimum distance is therefore $d_2 = 2^{m-1}$. The generator matrix EL_m for $RM(1,m)$ is then constructed from the generators $G_1 = EL_{m-1}$ and $G_2 = [1 \cdots 1]$ according to the recipe of Problem 4.1.5. Therefore, by that problem, we have

$$d_{min}(RM(1,m)) = \min(2d_1, d_2) = \min(2.2^{m-2}, 2^{m-1}) = 2^{m-1} ,$$

as claimed. □

(4.3.2) THEOREM. *The first order Reed-Muller code $RM(1,m)$ consists of a unique word of weight 0, namely $\mathbf{0}$, a unique word of weight 2^m, namely $\mathbf{1}$, and $2^{m+1}-2$ words of weight 2^{m-1}.*

PROOF. The last row of the generator matrix EL_m is $\mathbf{1}$; so $\mathbf{0}$ and $\mathbf{1}$ are the unique codewords of weight 0 and 2^m, respectively. By Theorem 4.3.1 the linear code $RM(1,m)$ has no codewords $\mathbf{c}$ of weight between 0 and 2^{m-1}, and so it also has no codewords $\mathbf{1}+\mathbf{c}$ of weight between 0 and 2^{m-1}. That is, it has no codewords of weight between 2^{m-1} and 2^m. Therefore all codewords other than $\mathbf{0}$ and $\mathbf{1}$ have weight exactly 2^{m-1}. □

(4.3.3) COROLLARY. *The dual of the binary Hamming code of redundancy m consists of a unique word of weight 0, namely $\mathbf{0}$, and 2^m-1 words of weight 2^{m-1}.*

PROOF. In recovering the dual Hamming code from $RM(1,m)$, we shorten the code by taking all codewords that begin with 0 and then delete that position. In particular the codeword $\mathbf{1}$ of $RM(1,m)$ does not survive. But by Theorem 4.3.2 all other nonzero codewords of $RM(1,m)$ have weight 2^{m-1}. As only zeros are deleted, all the nonzero codewords of the dual Hamming code also will have weight 2^{m-1}. □

equidistant codes

These dual Hamming codes are *equidistant codes* in that distinct codewords are at a fixed distance from each other, here 2^{m-1}. They satisfy the Plotkin bound 2.3.8 with equality. (The proof of the Plotkin bound as given in Problem 3.1.5 compares the minimum distance with the average distance between codewords. For an equidistant code these are the same.)

For a binary word $\mathbf{x} \in \mathbb{F}_2^n$, consider the corresponding word $\mathbf{x}^* \in \{+1,-1\}^n$ gotten by replacing each 0 by the real number $+1$ and each 1 by -1.

(4.3.4) LEMMA. *If $\mathbf{x},\mathbf{y} \in \mathbb{F}_2^n$, then as vectors of real numbers $\mathbf{x}^* \cdot \mathbf{y}^* = n - 2d_H(\mathbf{x},\mathbf{y})$. In particular if $\mathbf{x},\mathbf{y} \in \mathbb{F}_2^{2h}$ with $d_H(\mathbf{x},\mathbf{y}) = h$, then $\mathbf{x}^* \cdot \mathbf{y}^* = 0$.*

PROOF. The dot product of two ± 1 vectors is the number of places in which they are the same minus the number of places where they are different. Here that is $(n - d_H(\mathbf{x},\mathbf{y})) - d_H(\mathbf{x},\mathbf{y})$. □

Let $RM(1,m)^{\pm}$ be the code got by replacing each codeword $\mathbf{c}$ of $RM(1,m)$ with its ± 1 version $\mathbf{c}^*$. List the codewords of $RM(1,m)^{\pm}$ as $\mathbf{c}_1^*, \mathbf{c}_2^*, \ldots, \mathbf{c}_{2^{m+1}}^*$.

(4.3.5) LEMMA. *If $\mathbf{c}^* \in RM(1,m)^{\pm}$ then also $-\mathbf{c}^* \in RM(1,m)^{\pm}$. We have*

$$\begin{array}{rcll} \mathbf{c}_i^* \cdot \mathbf{c}_j^* & = & 2^m & \text{if } \mathbf{c}_i^* = \mathbf{c}_j^* \\ & = & -2^m & \text{if } \mathbf{c}_i^* = -\mathbf{c}_j^* \\ & = & 0 & \text{if } \mathbf{c}_i^* \neq \pm\mathbf{c}_j^* \ . \end{array}$$

PROOF. As $\mathbf{1} \in RM(1,m)$ we have $(\mathbf{1}+\mathbf{c})^* = -\mathbf{c}^* \in RM(1,m)^{\pm}$. By Theorem 4.3.2, if distinct $\mathbf{b},\mathbf{c} \in RM(1,m)$ with $\mathbf{b} \neq \mathbf{1}+\mathbf{c}$, then $d_H(\mathbf{b},\mathbf{c}) = 2^{m-1}$. The lemma follows from Lemma 4.3.4. □

We use this lemma as the basis of a decoding algorithm. When a vector $\mathbf{r}$ is received, calculate each of the dot products $\mathbf{r}\cdot\mathbf{c}_i^*$, for $i = 1,\ldots,2^{m+1}$. Then decode to that codeword $\mathbf{c}_j^*$ that maximizes the dot product.

In fact this can be done a little more efficiently. Arrange our listing of $RM(1,r)$ so that $\mathbf{c}_{i+2^m}^* = -\mathbf{c}_i^*$, for each $i = 1,\ldots,2^m$. That is, the second half of the list is just the negative of the first half. In decoding, we calculate only those dot products from the first half $\mathbf{r}\cdot\mathbf{c}_i^*$, for $i = 1,\ldots,2^m$, and select that j that maximizes the absolute value $|\mathbf{r}\cdot\mathbf{c}_j^*|$. The received word $\mathbf{r}$ is then decoded to $\mathbf{c}_j^*$ if $\mathbf{r}\cdot\mathbf{c}_j^*$ is positive and to $-\mathbf{c}_j^*$ if $\mathbf{r}\cdot\mathbf{c}_j^*$ is negative.

We organize this as *Hadamard transform decoding*. Set $n = 2^m$, and let H_n be the $n \times n$ matrix whose i^{th} row is the codeword $\mathbf{c}_i^*$. The dot products of Lemma 4.3.5 then give

Hadamard transform decoding

$$H_n H_n^{\top} = n\, I_{n\times n}\,,$$

since the negative of a row of H_n is never a row. Upon receiving the vector $\mathbf{r}$, we calculate its Hadamard transform $\widehat{\mathbf{r}} = H_n\mathbf{r}^{\top}$. If $\widehat{r}_j$ is that coordinate of $\widehat{\mathbf{r}}$ that has the largest absolute value, then we decode $\mathbf{r}$ to $\mathbf{c}_j^*$ in case $\widehat{r}_j > 0$ or to to $-\mathbf{c}_j^*$ in case $\widehat{r}_j < 0$.

An important aspect of Hadamard transform decoding is that it is a soft decision algorithm rather than a hard decision algorithm. We need not require that the received vector $\mathbf{r}$ have entries ± 1. Its entries can be arbitrary real numbers, and the algorithm still works without modification.

EXAMPLE. Consider the code $RM(1,3)^{\pm}$ which comes from $RM(1,3)$ of length $n = 2^3 = 8$ with generator matrix EL_3. Let

$$H_8 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix},$$

so $H_8 H_8^{\top} = 8\, I_{8\times 8}$. The codewords of $RM(1,3)^{\pm}$ are then the rows of H_8 and their negatives.

Suppose we receive the vector $\mathbf{r} = (1,1,-1,1,-1,-1,1,1)$. This has Hadamard transform $H_8\mathbf{r}^{\top} = (2,-2,2,-2,2,-2,6,2)$. The entry with largest absolute value is $\widehat{r}_7 = 6 > 0$, so we decode to

$$\mathbf{c}_7^* = (+1,+1,-1,-1,-1,-1,+1,+1)\,.$$

If next we receive $\mathbf{r} = (-.7,1,0,-.8,-.9,1,.9,-1)$, then

$$H_8\mathbf{r}^{\top} = (-.5,-.9,1.3,-6.3,-.5,-.9,.9,1.3)\,.$$

The entry with largest absolute value is $\widehat{r}_4 = -6.3 < 0$, so we decode to

$$-\mathbf{c}_4^* = (-1, +1, +1, -1, -1, +1, +1, -1)\,.$$

(4.3.6) PROBLEM. *Assume that you are using the code $RM(1,3)^{\pm}$ of the example. Use Hadamard transform decoding to decode the received word*

$$(.5\,,\ .4\,,\ -.6\,,\ .5\,,\ .6\,,\ -.3\,,\ .5\,,\ -.6\,)\ .$$

For any positive integer n, a ± 1 square matrix H_n of side n that satisfies

$$H_n H_n^{\top} = n\, I_{n\times n}$$

Hadamard matrix

is called a *Hadamard matrix*. If we take as a code the rows of H_n and their negatives, then Hadamard transform decoding will work exactly as described above. Such a code (or its $\{0,1\}$ counterpart) is called a *Hadamard code*.

Hadamard code

(4.3.7) PROBLEM. *Prove that a Hadamard matrix H_n must have $n = 1, 2$ or n a multiple of 4. (*REMARK. *It is a long-standing conjecture of combinatorial design theory that the converse of this problem is true: for each such n, there exists a Hadamard matrix.)*

Begin with a Hadamard code of length n. Choose those n codewords that start with $+1$, drop that position, and translate back to a $\{0,1\}$ code. The result is a binary code of length $n-1$ and size n which is equidistant of distance $n/2$. A code constructed in this fashion is a *shortened Hadamard code*. Starting with the matrix H_8 of the example above, we recover from $RM(1,3)$ and $RM(1,3)^{\pm}$ the $[7,3]$ dual Hamming code.

shortened Hadamard code

(4.3.8) PROBLEM. *Let h be a positive integer. Let C be a binary equidistant code of length $2h-1$ and size $2h$ with distance h.*

(*a*) *Prove that C is a shortened Hadamard code.*

(*b*) *Prove that C meets the Plotkin bound 2.3.8 with equality.*

Although any Hadamard matrix can be used to design a code that allows Hadamard transform decoding, there are certain advantages to be gained from using those matrices that come from Reed-Muller codes as described. The existence of a soft decision algorithm is good, but we hope to implement it as efficiently as possible. Consider decoding using the matrix H_8 of the example. Each decoding process requires 63 operations, 56 additions for calculating the 8 dot products and 7 comparisons among the answers. (By convention the operation of negation is considered to make no contribution.) Certain annoying repetitions are involved. For instance, both the second and the sixth rows of H_8 begin $+1, -1, +1, -1$; so the corresponding calculation $r_1 - r_2 + r_3 - r_4$ is made twice during the decoding process. Can this and other patterns within H_8 be exploited? The answer is "yes," and it is this fact that makes a matrix derived from $RM(1,m)$ a better choice than other Hadamard matrices with the same dimensions.

Let $H_1 = [1]$, a 1×1 Hadamard matrix, and define recursively a $2^{m+1} \times 2^{m+1}$ matrix in block form

$$H_{2^{m+1}} = \begin{bmatrix} +H_{2^m} & +H_{2^m} \\ +H_{2^m} & -H_{2^m} \end{bmatrix}.$$

Then

$$H_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix},$$

and

$$H_4 = \left[\begin{array}{cc|cc} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ \hline +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{array}\right] = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}.$$

The matrix H_8 is that of the example. This construction can be continued, for all m. The matrix H_{2^m} produced is a Hadamard matrix associated with $RM(1,m)^{\pm}$ and the Reed-Muller code $RM(1,m)$ whose generator matrix is EL_m. The recursive construction of H_{2^m} is related to that of EL_m and admits a streamlined implementation of decoding for $RM(1,m)$ and $RM(1,m)^{\pm}$, using the so-called Fast Hadamard Transform or FHT algorithm. For instance, FHT decoding of $RM(1,3)^{\pm}$ can be achieved with 31 operations rather than the 63 counted previously.

The Reed-Muller codes in general, and the code $RM(1,3)$ in particular, are important codes that often arise as constituents of larger codes. It is therefore worthwhile to have decoding algorithms that are as efficient as possible. Sun and Van Tilborg have given a soft decision algorithm for $RM(1,3)$ that is related to FHT decoding but only needs, on the average, 14 operations with worst-case performance of 17 operations.

Chapter 5

Generalized Reed-Solomon Codes

In 1960, I.S. Reed and G. Solomon introduced a family of error-correcting codes that are doubly blessed. The codes and their generalizations are useful in practice, and the mathematics that lies behind them is interesting. In the first section we give the basic properties and structure of the generalized Reed-Solomon codes, and in the second section we describe in detail one method of algebraic decoding that is quite efficient.

5.1 Basics

Let F be a field and choose nonzero elements $v_1, \ldots, v_n \in F$ and distinct elements $\alpha_1, \ldots, \alpha_n \in F$. Set $\mathbf{c} = (v_1, \ldots, v_n)$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. For $0 \leq k \leq n$ we define the *generalized Reed-Solomon codes* generalized Reed-Solomon codes

$$GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v}) = \{\ (v_1 f(\alpha_1), v_2 f(\alpha_2), \ldots, v_n f(\alpha_n)) \mid f(x) \in F[x]_k\ \}\ .$$

Here we write $F[x]_k$ for the set of polynomial in $F[x]$ of degree less than k, a vector space of dimension k over F. For fixed n, $\boldsymbol{\alpha}$, and $\mathbf{v}$, the various GRS codes enjoy the nice embedding property $GRS_{n,k-1}(\boldsymbol{\alpha}, \mathbf{v}) \leq GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$.

If $f(x)$ is a polynomial, then we shall usually write $\mathbf{f}$ for its associated codeword. This codeword also depends upon $\boldsymbol{\alpha}$ and $\mathbf{v}$; so at times we prefer to write unambiguously

$$\mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(f(x)) = (v_1 f(\alpha_1), v_2 f(\alpha_2), \ldots, v_n f(\alpha_n))\,,$$

indicating that the codeword $\mathbf{f} = \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(f(x))$ arises from evaluating the polynomial $f(x)$ at $\boldsymbol{\alpha}$ and scaling by $\mathbf{v}$.

(5.1.1) THEOREM. *$GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is an $[n, k]$ linear code over F with length $n \leq |F|$. We have $d_{min} = n - k + 1$ provided $k \neq 0$. In particular, GRS codes are MDS codes.*

PROOF. As by definition the entries in α are distinct, we must have $n \leq |F|$. If $a \in F$ and $f(x), g(x) \in F[x]_k$, then $af(x) + g(x)$ is also in $F[x]_k$; and

$$\mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(af(x) + g(x)) = a\,\mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(f(x)) + \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(g(x)) = a\mathbf{f} + \mathbf{g}\,.$$

Therefore $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is linear of length n over F.

Let $f(x), g(x) \in F[x]_k$ be distinct polynomials. Set $h(x) = f(x) - g(x) \neq 0$, also of degree less than k. Then $\mathbf{h} = \mathbf{f} - \mathbf{g}$ and $w_H(\mathbf{h}) = d_H(\mathbf{f}, \mathbf{g})$. But the weight of $\mathbf{h}$ is n minus the number of 0's in $\mathbf{h}$. As all the v_i are nonzero, this equals n minus the number of roots that $h(x)$ has among $\{\alpha_1, \ldots, \alpha_n\}$. As $h(x)$ has at most $k-1$ roots by Proposition A.2.10, the weight of $\mathbf{h}$ is at least $n - (k-1) = n - k + 1$. Therefore $d_{min} \geq n - k + 1$, and we get equality from the Singleton bound 3.1.14. (Alternatively, $h(x) = \prod_{i=1}^{k-1}(x - \alpha_i)$ produces a codeword $\mathbf{h}$ of weight $n - k + 1$.)

The argument of the previous paragraph also shows that distinct polynomials $f(x), g(x)$ of $F[x]_k$ give distinct codewords. Therefore the code contains $|F|^k$ codewords and has dimension k. □

The vector $\mathbf{v}$ plays little role here, and its uses will be more apparent later. At present, it serves to make sure that any code that is monomially equivalent to a *GRS* code is itself a *GRS* code.

Let us now find a generator matrix for $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$. The argument of Theorem 5.1.1 makes it clear that any basis $f_1(x), \ldots, f_k(x)$ of $F[x]_k$ gives rise to a basis $\mathbf{f}_1, \ldots, \mathbf{f}_k$ of the code. A particularly nice polynomial basis is the set of monomials $1, x, \ldots, x^i, \ldots, x^{k-1}$. The corresponding generator matrix, whose i^{th} row (numbering rows from 0 to $k-1$) is

$$\mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(x^i) = (v_1\alpha_1^i, \ldots, v_j\alpha_j^i, \ldots, v_n\alpha_n^i)\,,$$

canonical generator matrix

is the *canonical generator matrix* for $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$:

$$\begin{bmatrix} v_1 & v_2 & \ldots & v_j & \ldots & v_n \\ v_1\alpha_1 & v_2\alpha_2 & \ldots & v_j\alpha_j & \ldots & v_n\alpha_n \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_1\alpha_1^i & v_2\alpha_2^i & \ldots & v_j\alpha_j^i & \ldots & v_n\alpha_n^i \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_1\alpha_1^{k-1} & v_2\alpha_2^{k-1} & \ldots & v_j\alpha_j^{k-1} & \ldots & v_n\alpha_n^{k-1} \end{bmatrix}$$

(5.1.2) PROBLEM. *Consider the code $C = GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$, and assume that all the entries of the vector $\boldsymbol{\alpha}$ are nonzero. If*

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ \ldots\ , \alpha_n)\ ,$$

define

$$\boldsymbol{\beta} = (\alpha_1^{-1}, \alpha_2^{-1}, \ \ldots\ , \alpha_n^{-1})\ .$$

Find a vector $\mathbf{w}$ such that $C = GRS_{n,k}(\beta, \mathbf{w})$.

(5.1.3) PROBLEM. *(a) Consider the code $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ over F. Let a, c be nonzero elements of F, and let $\mathbf{b}$ be the vector of F^n all of whose entries are equal to $b \in F$. Prove that*

$$GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v}) = GRS_{n,k}(a\boldsymbol{\alpha} + \mathbf{b}, c\mathbf{v}) .$$

(b) If $n < |F|$, prove that there is an $\boldsymbol{\alpha}'$ with no entries equal to 0 and

$$GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v}) = GRS_{n,k}(\boldsymbol{\alpha}', \mathbf{v}) .$$

(5.1.4) PROBLEM. *Consider the code E, which will be linear of length $n+1$ and dimension k, whose generator matrix results from adding a new column to the canonical generator matrix for $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$:*

$$\begin{bmatrix} v_1 & v_2 & \dots & v_j & \dots & v_n & 0 \\ v_1\alpha_1 & v_2\alpha_2 & \dots & v_j\alpha_j & \dots & v_n\alpha_n & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ v_1\alpha_1^i & v_2\alpha_2^i & \dots & v_j\alpha_j^i & \dots & v_n\alpha_n^i & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ v_1\alpha_1^{k-2} & v_2\alpha_2^{k-2} & \dots & v_j\alpha_j^{k-2} & \dots & v_n\alpha_n^{k-2} & 0 \\ v_1\alpha_1^{k-1} & v_2\alpha_2^{k-1} & \dots & v_j\alpha_j^{k-1} & \dots & v_n\alpha_n^{k-1} & 1 \end{bmatrix}$$

Prove that $d_{min}(E) = n - k + 2$.

REMARK. *As $n - k + 2 = (n + 1) - k + 1$, this shows that the code E satisfies the Singleton Bound with equality and so is maximum distance separable (MDS), just as all GRS codes are.*

It is extremely profitable to think of Theorem 5.1.1 again in terms of polynomial interpolation:

> Any polynomial of degree less than k is uniquely determined by its values at k (or more) distinct points.

Here, any codeword with as many as k entries equal to 0 corresponds to a polynomial of degree less than k whose values match the 0 polynomial in k points and so must itself be the 0 polynomial.

Given any n-tuple $\mathbf{f}$, we can easily reconstruct the polynomial $f(x)$ that produces $\mathbf{f} = \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(f(x))$. We first introduce some notation. Set

$$L(x) = \prod_{i=1}^{n}(x - \alpha_i)$$

and

$$L_i(x) = L(x)/(x - \alpha_i) = \prod_{j \neq i}(x - \alpha_j) .$$

The polynomials $L(x)$ and $L_i(x)$ are monic of degrees n and $n-1$, respectively. The vector $\mathbf{f}$ has i^{th} coordinate $v_i f(\alpha_i)$, so we have enough information to calculate, using the Lagrange interpolation formula A.2.11,

$$f(x) = \sum_{i=1}^{n} \frac{L_i(x)}{L_i(\alpha_i)} f(\alpha_i) .$$

The coefficients $L_i(\alpha_i)$ (always nonzero) are often easy to compute.

(5.1.5) PROBLEM. *(a) Prove that $L_i(\alpha_i) = L'(\alpha_i)$, where $L'(x)$ is the formal derivative of $L(x)$ as defined in Problem A.2.26.*

(b) If $n = |F|$ and $\{\alpha_1, \ldots, \alpha_n\} = F$, then $L_i(\alpha_i) = -1$, for all i.

(c) If $\{\alpha_1, \ldots, \alpha_n\}$ is composed of n roots of $x^n - 1$ in F, then $L_i(\alpha_i) = n\alpha_i^{-1}$ $(\neq 0)$. In particular, if $n = |F| - 1$ and $\{\alpha_1, \ldots, \alpha_n\} = F - \{0\}$, then $L_i(\alpha_i) = -\alpha_i^{-1}$ (hence $\alpha_i^{-1} L_i(\alpha_i)^{-1} = -1$).

The polynomial $f(x)$ has degree less than k, while the interpolation polynomial of the righthand side above has apparent degree $n - 1$. The resolution of this confusion allows us to find the dual of a *GRS* code easily.

(5.1.6) THEOREM. *We have*

$$GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})^{\perp} = GRS_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u}),$$

where $\mathbf{u} = (u_1, \ldots, u_n)$ with $u_i^{-1} = v_i \prod_{j \neq i}(\alpha_i - \alpha_j)$.

PROOF. By definition $u_i = v_i^{-1} L_i(\alpha_i)^{-1}$.

We prove that every $\mathbf{f}$ in $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ has dot product 0 with every $\mathbf{g}$ in $GRS_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u})$, from which the result is immediate. Let $\mathbf{f} = \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{v}}(f(x))$ and $\mathbf{g} = \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{u}}(g(x))$. The polynomial $f(x)$ has degree less than k while $g(x)$ has degree less than $n - k$. Therefore their product $f(x)g(x)$ has degree at most $n - 2$. By Lagrange interpolation A.2.11 we have

$$f(x)g(x) = \sum_{i=1}^{n} \frac{L_i(x)}{L_i(\alpha_i)} f(\alpha_i) g(\alpha_i) \ .$$

Equating the coefficient of x^{n-1} from the two sides gives:

$$\begin{aligned} 0 &= \sum_{i=1}^{n} \frac{1}{L_i(\alpha_i)} f(\alpha_i) g(\alpha_i) \\ &= \sum_{i=1}^{n} (v_i f(\alpha_i)) \left(\frac{v_i^{-1}}{L_i(\alpha_i)} g(\alpha_i) \right) \\ &= \sum_{i=1}^{n} (v_i f(\alpha_i))(u_i g(\alpha_i)) \\ &= \mathbf{f} \cdot \mathbf{g} \, , \end{aligned}$$

as required. □

The ability in the class of *GRS* codes to choose different vectors $\mathbf{v}$ to accompany a fixed $\boldsymbol{\alpha}$ has been helpful here.

Of course, to specify $\mathbf{f}$ as a codeword in $C = GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ we do not need to check it against every $\mathbf{g}$ of $C^{\perp} = GRS_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u})$. It is enough to consider a basis of $C^{\perp}$, a nice one being the rows of the canonical generator matrix for

$C^\perp$, a check matrix for C. Our introduction of GRS codes such as C essentially defines them via their canonical generator matrices. As we have seen before, describing a linear code instead in terms of a check matrix can be fruitful. In particular this opens the possibility of syndrome decoding.

Set $r = n - k$, and let $\mathbf{c} = (c_1, \ldots, c_n) \in F^n$. Then

$$\begin{aligned} \mathbf{c} \in C \quad &\text{iff} \quad 0 = \mathbf{c} \cdot \mathbf{ev}_{\boldsymbol{\alpha},\mathbf{u}}(x^j), \quad \text{for } 0 \le j \le r-1 \\ &\text{iff} \quad 0 = \sum_{i=1}^{n} c_i u_i \alpha_i^j, \quad \text{for } 0 \le j \le r-1\,. \end{aligned}$$

We rewrite these r equations as a single equation in the polynomial ring $F[z]$ in a new indeterminate z. The vector $\mathbf{c}$ is in C if and only if in $F[z]$ we have

$$\begin{aligned} 0 &= \sum_{j=0}^{r-1} \Big(\sum_{i=1}^{n} c_i u_i \alpha_i^j \Big) z^j \\ &= \sum_{i=1}^{n} c_i u_i \Big(\sum_{j=0}^{r-1} (\alpha_i z)^j \Big) \end{aligned}$$

The polynomials $1 - \alpha z$ and z^r are relatively prime, so it is possible to invert $1 - \alpha z$ in the ring $F[z] \pmod{z^r}$. Indeed

$$1/(1 - \alpha z) = \sum_{j=0}^{r-1} (\alpha z)^j \pmod{z^r}\,,$$

a truncation of the usual geometric series expansion. We are left with:

(5.1.7) THEOREM. *(GOPPA FORMULATION FOR GRS CODES.) The generalized Reed-Solomon code $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ over F is equal to the set of all n-tuples $\mathbf{c} = (c_1, c_2, \ldots, c_n) \in F^n$, such that*

$$\sum_{i=1}^{n} \frac{c_i u_i}{1 - \alpha_i z} = 0 \pmod{z^r}\,,$$

where $r = n - k$ and $u_i^{-1} = v_i \prod_{j \ne i} (\alpha_i - \alpha_j)$. □

This interpretation of GRS codes has two main values. First, it is open to a great deal of generalization, as we shall later see. Second, it suggests a practical method for the decoding of GRS codes, the topic of the next section.

5.2 Decoding GRS codes

As GRS codes are MDS, they can be decoded using threshold decoding as in Problem 3.2.4. We now present an efficient and more specific algorithm for decoding the dual of $GRS_{n,r}(\boldsymbol{\alpha}, \mathbf{u})$, starting from the Goppa formulation.

Suppose $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ is transmitted, and $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ is received, so that the error vector $\mathbf{e} = (e_1, e_2, \ldots, e_n)$ has been introduced; $\mathbf{p} = \mathbf{c} + \mathbf{e}$. We calculate the *syndrome polynomial* of $\mathbf{p}$:

syndrome polynomial

$$S_{\mathbf{p}}(z) = \sum_{i=1}^{n} \frac{p_i u_i}{1 - \alpha_i z} \pmod{z^r} .$$

Then it is easily seen that

$$S_{\mathbf{p}}(z) = S_{\mathbf{c}}(z) + S_{\mathbf{e}}(z) \pmod{z^r},$$

whence, by the Goppa formulation of Theorem 5.1.7,

$$S_{\mathbf{p}}(z) = S_{\mathbf{e}}(z) \pmod{z^r}.$$

Let B be the set of error locations:

$$B = \{i \mid e_i \neq 0\}.$$

Then we have the syndrome polynomial

$$S_{\mathbf{p}}(z) = S_{\mathbf{e}}(z) = \sum_{b \in B} \frac{e_b u_b}{1 - \alpha_b z} \pmod{z^r}.$$

We now drop the subscripts and write $S(z)$ for the syndrome polynomial.

Key Equation

Clear denominators to find the *Key Equation*:

$$\sigma(z)S(z) = \omega(z) \pmod{z^r},$$

where

$$\sigma(z) = \sigma_{\mathbf{e}}(z) = \prod_{b \in B}(1 - \alpha_b z)$$

and

$$\omega(z) = \omega_{\mathbf{e}}(z) = \sum_{b \in B} e_b u_b \Big(\prod_{a \in B, a \neq b} (1 - \alpha_a z) \Big) .$$

error locator
error evaluator

(Empty products are taken as 1.) The polynomial $\sigma(z)$ is called the *error locator* polynomial, and the polynomial $\omega(z)$ is the *error evaluator* polynomial.

The names are justifiable. Given the polynomials $\sigma(z) = \sigma_{\mathbf{e}}(z)$ and $\omega(z) = \omega_{\mathbf{e}}(z)$, we can reconstruct the error vector $\mathbf{e}$. Assume for the moment that none of the α_i are equal to 0 (although similar results are true when some α_i is 0) . Then:

$$B = \{\, b \mid \sigma(\alpha_b^{-1}) = 0 \,\}$$

and, for each $b \in B$,

$$e_b = \frac{-\alpha_b \omega(\alpha_b^{-1})}{u_b \sigma'(\alpha_b^{-1})} ,$$

where $\sigma'(z)$ is the formal derivative of $\sigma(z)$ (see Problem A.2.26). In fact the polynomials $\sigma(z)$ and $\omega(z)$ determine the error vector even when some α_i equals 0.

If the error vector $\mathbf{e} \neq \mathbf{0}$ has weight at most $r/2$ $(= (d_{min} - 1)/2)$, then relative to the syndrome polynomial $S_{\mathbf{e}}(z) = S(z)$ the pair of polynomials $\sigma_{\mathbf{e}}(z) = \sigma(z)$ and $\omega_{\mathbf{e}}(z) = \omega(z)$ has the three properties by which it is characterized in the next theorem. Indeed (1) is just the Key Equation. Property (2) is a consequence of the assumption on error weight and the definitions of the polynomials $\sigma(z)$ and $\omega(z)$. For (3) we have $\sigma(0) = 1$ trivially. As $\sigma(z)$ is a product of linear factors, either $\gcd(\sigma(z), \omega(z)) = 1$ or the two polynomials have a common root. But for each root α_b^{-1} of $\sigma(z)$ we have $0 \neq \omega(\alpha_b^{-1})$, a factor of $e_b \neq 0$.

Our decoding method solves the Key Equation and so finds the error vector $\mathbf{e}$ as above. The following theorem provides us with a characterization of the solutions we seek.

(5.2.1) THEOREM. *Given r and $S(z)$ there is at most one pair of polynomials $\sigma(z), \omega(z)$ satisfying:*
(1) $\sigma(z)S(z) = \omega(z) \pmod{z^r}$;
(2) $\deg(\sigma(z)) \leq r/2$ *and* $\deg(\omega(z)) < r/2$;
(3) $\gcd(\sigma(z), \omega(z)) = 1$ *and* $\sigma(0) = 1$.

In fact we prove something slightly stronger.

(5.2.2) PROPOSITION. *Assume that $\sigma(z), \omega(z)$ satisfy* (1)-(3) *of Theorem 5.2.1 and that $\sigma_1(z), \omega_1(z)$ satisfy* (1) *and* (2). *Then there is a polynomial $\mu(z)$ with $\sigma_1(z) = \mu(z)\sigma(z)$ and $\omega_1(z) = \mu(z)\omega(z)$.*

PROOF. From (1)

$$\sigma(z)\omega_1(z) = \sigma(z)\sigma_1(z)S(z) = \sigma_1(z)\omega(z) \pmod{z^r};$$

so

$$\sigma(z)\omega_1(z) - \sigma_1(z)\omega(z) = 0 \pmod{z^r}.$$

But by (2) the lefthand side of this equation has degree less than r. Therefore

$$\sigma(z)\omega_1(z) = \sigma_1(z)\omega(z).$$

From (3) we have $\gcd(\sigma(z), \omega(z)) = 1$, so by Lemma A.2.20 $\sigma(z)$ divides $\sigma_1(z)$. Set $\sigma_1(z) = \mu(z)\sigma(z)$. Then

$$\sigma(z)\omega_1(z) = \sigma_1(z)\omega(z) = \sigma(z)\mu(z)\omega(z).$$

The polynomial $\sigma(z)$ is nonzero since $\sigma(0) = 1$; so by cancellation $\omega_1(z) = \mu(z)\omega(z)$, as desired. □

PROOF OF THEOREM 5.2.1.

Any second such pair has

$$\sigma_1(z) = \mu(z)\sigma(z) \text{ and } \omega_1(z) = \mu(z)\omega(z)$$

by the proposition. So $\mu(z)$ divides $\gcd(\sigma_1(z), \omega_1(z))$ which is 1 by (3). Therefore $\mu(z) = \mu$ is a constant. Indeed

$$1 = \sigma_1(0) = \mu(0)\sigma(0) = \mu \cdot 1 = \mu\,.$$

Thus $\sigma_1(z) = \mu(z)\sigma(z) = \sigma(z)$ and $\omega_1(z) = \mu(z)\omega(z) = \omega(z)$. □

Using the characterization of Theorem 5.2.1 we now verify a method of solving the Key Equation with the Euclidean algorithm, as presented in Section A.3.1 of the appendix.

(5.2.3) THEOREM. *(*DECODING *GRS* USING THE EUCLIDEAN ALGORITHM.*) Consider the code $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ over F, and set $r = n - k$. Given a syndrome polynomial $S(z)$ (of degree less than r), the following algorithm halts, producing polynomials $\tilde{\sigma}(z)$ and $\tilde{\omega}(z)$:*

Set $a(z) = z^r$ and $b(z) = S(z)$.
Step through the Euclidean Algorithm A.3.1
until at Step j, $\deg(r_j(z)) < r/2$.
Set $\tilde{\sigma}(z) = t_j(z)$
and $\tilde{\omega}(z) = r_j(z)$.

If there is an error word $\mathbf{e}$ of weight at most $r/2 = (d_{min} - 1)/2$ with $S_{\mathbf{e}}(z) = S(z)$, then $\widehat{\sigma}(z) = \tilde{\sigma}(0)^{-1}\tilde{\sigma}(z)$ and $\widehat{\omega}(z) = \tilde{\sigma}(0)^{-1}\tilde{\omega}(z)$ are the error locator and evaluator polynomials for $\mathbf{e}$.

PROOF. It is the goal of the Euclidean algorithm to decrease the degree of $r_j(z)$ at each step, so the algorithm is guaranteed to halt.

Now assume that $S(z) = S_{\mathbf{e}}(z)$ with $w_H(\mathbf{e}) \leq r/2$. Therefore the error locator and evaluator pair $\sigma(z) = \sigma_{\mathbf{e}}(z)$ and $\omega(z) = \omega_{\mathbf{e}}(z)$ satisfies (1), (2), and (3) of Theorem 5.2.1. We first check that, for the j defined, the pair $t_j(z)$ and $r_j(z)$ satisfies (1) and (2).

Requirement (1) is just the Key Equation and is satisfied at each step of the Euclidean algorithm since always

$$E_j\colon \quad r_j(z) = s_j(z)z^r + t_j(z)S(z).$$

For (2), our choice of j gives $\deg(r_j(z)) < r/2$ and also $\deg(r_{j-1}(z)) \geq r/2$. Therefore, from Problem A.3.5,

$$\begin{aligned} \deg(t_j(z)) + r/2 &\leq \deg(t_j(z)) + \deg(r_{j-1}(z)) \\ &= \deg(a(z)) = \deg(z^r) = r\,. \end{aligned}$$

Hence $\deg(t_j(z)) \leq r/2$, giving (2).

By Proposition 5.2.2 there is a polynomial $\mu(z)$ with

$$t_j(z) = \mu(z)\sigma(z) \text{ and } r_j(z) = \mu(z)\omega(z)\,.$$

Here $\mu(z)$ is not the zero polynomial by Lemma A.3.3(a).

If we substitute for $t_j(z)$ and $r_j(z)$ in equation E_j we have

$$s_j(z)z^r + (\mu(z)\sigma(z))S(z) = \mu(z)\omega(z)\,,$$

which becomes

$$\mu(z)\Big(\omega(z) - \sigma(z)S(z)\Big) = s_j(z)z^r\,.$$

By the Key Equation, the parenthetical expression on the left is $p(z)z^r$, for some $p(z)$; so we are left with $\mu(z)p(z)z^r = s_j(z)z^r$ or

$$\mu(z)p(z) = s_j(z)\,.$$

Thus $\mu(z)$ divides $\gcd(t_j(z), s_j(z))$, which is 1 by Corollary A.3.4.

We conclude that $\mu(z) = \mu$ is a nonzero constant function. Furthermore

$$t_j(0) = \mu(0)\sigma(0) = \mu\,;$$

so

$$\sigma(z) = t_j(0)^{-1}t_j(z) \text{ and } \omega(z) = t_j(0)^{-1}r_j(z)\,,$$

as desired. □

When this algorithm is used, decoding default occurs when $\widehat{\sigma}(z)$ does not split into linear factors whose roots are inverses of entries in α with multiplicity 1. (Here we assume that none of the α_i are 0.) That is, the number of roots of $\widehat{\sigma}(z)$ among the α_i^{-1} must be equal to the degree of $\widehat{\sigma}(z)$. If this is not the case, then we have detected errors that we are not able to correct. Another instance of decoder default occurs when $t_j(0) = 0$, so the final division to determine $\widehat{\sigma}(z)$ can not be made.

If $t_j(0) \neq 0$ and $\widehat{\sigma}(z)$ does split as described, then we can go on to evaluate errors at each of the located positions and find a vector of weight at most $r/2$ with our original syndrome. In this case we have either decoded correctly, or we had more than $r/2$ errors and have made a decoding error. (We need not worry about division by 0 in evaluating errors, since this can only happen if $\widehat{\sigma}(z)$ has roots of multiplicity greater than one; see Problem A.2.27.)

Assume now that r is even or that $\boldsymbol{\alpha}$ has weight n. Then this algorithm only produces error vectors of weight $r/2$ or less. In particular if more than $r/2$ errors occur then we will have a decoding default or a decoder error. Suppose that we have found polynomials $\widehat{\sigma}(z)$ and $\widehat{\omega}(z)$ that allow us to calculate a candidate error vector $\mathbf{e}$ of weight at most $r/2$. It follows from Lagrange interpolation A.2.11 that $\widehat{\sigma}(z) = \sigma_{\mathbf{e}}(z)$ and $\widehat{\omega}(z) = \omega_{\mathbf{e}}(z)$. Also since $\sigma(z)$ is invertible modulo z^r, we can solve the Key Equation to find that $S(z) = S_{\mathbf{e}}(z)$. Therefore the received vector is within a sphere of radius $r/2$ around a codeword and is decoded to that codeword. That is, under these conditions Euclidean algorithm decoding as given in Theorem 5.2.3 is an explicit implementation of the decoding algorithm $\mathbf{SS}_{r/2}$.

EXAMPLE. Consider the code $C = GRS_{6,2}(\boldsymbol{\alpha}, \mathbf{v})$ over the field $\mathbb{F}_7$ of integers modulo 7, where

$$\boldsymbol{\alpha} = (2, 4, 6, 1, 3, 5)$$

and

$$\mathbf{v} = (1, 1, 1, 1, 1, 1)\,.$$

First calculate a vector $\mathbf{u}$ for which $C^\perp = GRS_{6,4}(\boldsymbol{\alpha}, \mathbf{u})$. Starting with

$$L(x) = (x-2)(x-4)(x-6)(x-1)(x-3)(x-5)$$

we find:

$$\begin{array}{lccccccccc}
L_1(2) = & & (-2) & (-4) & (1) & (-1) & (-3) & = & 24 & = 3 \\
L_2(4) = & (2) & & (-2) & (3) & (1) & (-1) & = & 12 & = 5 \\
L_3(6) = & (4) & (2) & & (5) & (3) & (1) & = & 120 & = 1 \\
L_4(1) = & (-1) & (-3) & (-5) & & (-2) & (-4) & = & -120 & = 6 \\
L_5(3) = & (1) & (-1) & (-3) & (2) & & (-2) & = & -12 & = 2 \\
L_6(5) = & (3) & (1) & (-1) & (4) & (2) & & = & -24 & = 4
\end{array}$$

(Notice that these values could have been found easily using Problem 5.1.5(c).) Now $u_i = (v_i L_i(\alpha_i))^{-1} = L_i(\alpha_i)^{-1}$ since $v_i = 1$; so

$$\mathbf{u} = (5, 3, 1, 6, 4, 2)\,.$$

Next calculate the syndrome polynomial of an arbitrary received vector

$$\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6)\,.$$

In our example $r = 6 - 2 = 4$.

$$S_{\mathbf{p}}(z) = \frac{5\cdot p_1}{1-2z} + \frac{3\cdot p_2}{1-4z} + \frac{1\cdot p_3}{1-6z} + \frac{6\cdot p_4}{1-1z} + \frac{4\cdot p_5}{1-3z} + \frac{2\cdot p_6}{1-5z} \pmod{z^4}$$

$$\begin{array}{rrrrrr}
= & 5p_1(& 1 & +2z & +4z^2 & +z^3) \\
 & +3p_2(& 1 & +4z & +2z^2 & +z^3) \\
 & +p_3(& 1 & +6z & +z^2 & +6z^3) \\
 & +6p_4(& 1 & +z & +z^2 & +z^3) \\
 & +4p_5(& 1 & +3z & +2z^2 & +6z^3) \\
 & +2p_6(& 1 & +5z & +4z^2 & +6z^3)
\end{array} \pmod{z^4}$$

$$\begin{array}{rrrrrr}
= & p_1(& 5 & +3z & +6z^2 & +5z^3) \\
 & +p_2(& 3 & +5z & +6z^2 & +3z^3) \\
 & +p_3(& 1 & +6z & +z^2 & +6z^3) \\
 & +p_4(& 6 & +6z & +6z^2 & +6z^3) \\
 & +p_5(& 4 & +5z & +z^2 & +3z^3) \\
 & +p_6(& 2 & +3z & +z^2 & +5z^3)
\end{array} \pmod{z^4}\,.$$

Notice that this calculation amounts to finding the canonical check matrix for the code.

We now use the algorithm of Theorem 5.2.3 to decode the received vector

$$\mathbf{p} = (1, 3, 6, 5, 4, 2)\,.$$

We have the syndrome polynomial

$$S(z) = \frac{5\cdot 1}{1-2z} + \frac{3\cdot 3}{1-4z} + \frac{1\cdot 6}{1-6z} + \frac{6\cdot 5}{1-1z} + \frac{4\cdot 4}{1-3z} + \frac{2\cdot 2}{1-5z} \pmod{z^4}$$

$$\begin{aligned} = \quad & 1(\ 5 \ +3z \ +6z^2 \ +5z^3) \\ & +3(\ 3 \ +5z \ +6z^2 \ +3z^3) \\ & +6(\ 1 \ +6z \ +z^2 \ +6z^3) \\ & +5(\ 6 \ +6z \ +6z^2 \ +6z^3) \\ & +4(\ 4 \ +5z \ +z^2 \ +3z^3) \\ & +2(\ 2 \ +3z \ +z^2 \ +5z^3) \end{aligned} \pmod{z^4} .$$

$$= 5z + 3z^2 + 4z^3 \pmod{z^4} .$$

The algorithm now requires that, starting with initial conditions

$$a(z) = z^4 \text{ and } b(z) = 4z^3 + 3z^2 + 5z\,,$$

we step through the Euclidean Algorithm until at Step j we first have $\deg(r_j(z)) < r/2 = 2$.

This is precisely the Euclidean Algorithm example done in the appendix. At Step 2. we have the first occurrence of a remainder term with degree less than 2; we have $r_2(z) = 6z$. We also have $t_2(z) = 3z^2+6z+4$, so $t_2(0)^{-1} = 4^{-1} = 2$. Therefore we have error locator and evaluator polynomials:

$$\sigma(z) = t_2(0)^{-1}t_2(z) = 2(3z^2 + 6z + 4) = 6z^2 + 5z + 1$$

$$\omega(z) = t_2(0)^{-1}r_2(z) = 2(6z) = 5z\ .$$

The error locations are those in $B = \{\, b \,|\, \sigma(\alpha_b^{-1}) = 0 \,\}$; so to find the error locations, we must extract the roots of $\sigma(z)$. As $\mathbb{F}_7$ does not have characteristic 2, we can use the usual quadratic formula and find that the roots are

$$2\,,\ 3 = \frac{-5 \pm \sqrt{25 - 4\cdot 6}}{2\cdot 6}\ .$$

Now $2^{-1} = 4 = \alpha_2$ and $3^{-1} = 5 = \alpha_6$, so $B = \{2, 6\}$.

An error value e_b is given by

$$e_b = \frac{-\alpha_b \omega(\alpha_b^{-1})}{u_b \sigma'(\alpha_b^{-1})}\ ,$$

where $\sigma'(z) = 5z + 5$. Thus $e_2 = \frac{-4\cdot 10}{3\cdot 15} = 3$ and $e_6 = \frac{-5\cdot 15}{2\cdot 20} = 6$.

We have thus found

$$\mathbf{e} = (0, 3, 0, 0, 0, 6)\,,$$

so we decode the received word $\mathbf{p}$ to

$$\mathbf{c} = \mathbf{p} - \mathbf{e} = (1, 3, 6, 5, 4, 2) - (0, 3, 0, 0, 0, 6) = (1, 0, 6, 5, 4, 3)\ .$$

In the example we have been able to use the quadratic formula to calculate the roots of $\sigma(z)$ and so find the error locations. This will not always be possible. There may be more than 2 errors. In any case, the quadratic formula involves division by 2 and so is not valid when the characteristic of the field F is 2, one

Chien search

of the most interesting cases. A method that is often used is the substitution, one-by-one, of all field elements into $\sigma(z)$, a *Chien search*. Although this lacks subtlety, it is manageable when the field is not too big. There do not seem to be general alternatives that are good and simple.

(5.2.4) PROBLEM. *Consider the* $GRS_{8,4}(\boldsymbol{\alpha}, \mathbf{v})$ *code* C *over* $\mathbb{F}_{13}$ *with*

$$\mathbf{v} = (1, 1, 1, 1, 1, 1, 1, 1)$$

$$\boldsymbol{\alpha} = (1, 4, 3, 12, 9, 10, 5, 8) \ .$$

(*a*) *Give* $n, k, \boldsymbol{\beta}, \mathbf{u}$ *with* $C^{\perp} = GRS_{n,k}(\boldsymbol{\beta}, \mathbf{u})$.

(*b*) *When transmitting with* C*, assume that the vector*

$$\mathbf{p} = (0, 0, 0, 0, 0, 0, 3, 5) \ .$$

is received. Use the Euclidean algorithm to find an error vector $\mathbf{e}$ *and a decoded codeword* $\mathbf{c}$*. (The answers should be obvious. Use the question to check your understanding of the process.)*

(*c*) *When transmitting with* C*, assume that the vector*

$$\mathbf{p} = (3, 6, 0, 4, 0, 5, 0, 12)$$

is received. Use the Euclidean algorithm to find an error vector $\mathbf{e}$ *and a decoded codeword* $\mathbf{c}$*.*

(5.2.5) PROBLEM. *Consider the* $GRS_{10,4}(\boldsymbol{\alpha}, \mathbf{v})$ *code* C *over* $\mathbb{F}_{13}$ *with*

$$\mathbf{v} = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$$

$$\boldsymbol{\alpha} = (1, 2, 3, 4, 6, 7, 9, 10, 11, 12) \ .$$

(*a*) *Give* $n, k, \boldsymbol{\beta}, \mathbf{u}$ *with* $C^{\perp} = GRS_{n,k}(\boldsymbol{\beta}, \mathbf{u})$.
*(*HINT: $\mathbf{u} = (*, *, 9, 10, 12, 1, 3, 4, *, *)$.*)*

(*b*) *When transmitting with* C*, assume that the vector*

$$\mathbf{p} = (4, 5, 6, 0, 0, 0, 0, 0, 0, 0) \ .$$

is received. Use the Euclidean algorithm to find an error vector $\mathbf{e}$ *and a decoded codeword* $\mathbf{c}$*. (The answers should be obvious. Use the question to check your understanding of the process.)*

(*c*) *When transmitting with* C*, assume that the vector*

$$\mathbf{p} = (3, 1, 0, 0, 0, 0, 0, 5, 7, 12) \ .$$

is received. Use the Euclidean algorithm to find an error vector $\mathbf{e}$ *and a decoded codeword* $\mathbf{c}$*.*

(5.2.6) PROBLEM. *Let the field $\mathbb{F}_8$ be given as polynomials of degree at most 2 in α, a root of the primitive polynomial $x^3+x+1 \in \mathbb{F}_2[x]$. Consider the code $C = GRS_{7,3}(\boldsymbol{\alpha}, \mathbf{v})$ over $\mathbb{F}_8$ with*

$$\boldsymbol{\alpha} = \mathbf{v} = (1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6)\,.$$

By Problem 5.1.5(c) we have $C^\perp = GRS_{7,4}(\boldsymbol{\alpha}, \mathbf{u})$ for $\mathbf{u} = (1,1,1,1,1,1,1)$.

When transmitting with C, assume that the vector

$$\mathbf{p} = (0, \alpha^5, 0, 1, \alpha^6, 0, 1)$$

is received. Use the Euclidean Algorithm to find an error vector $\mathbf{e}$ and a decoded codeword $\mathbf{c}$.

Chapter 6

Modifying Codes

If one code is in some sense good, then we can hope to find from it similar and related codes that are also good. In this chapter we discuss some elementary methods for modifying a code in order to find new codes. In two further sections we discuss special cases related to generalized Reed-Solomon codes.

6.1 Six basic techniques

A code C has three fundamental parameters — its length n, its dimension k, and its redundancy $r = n - k$. Each of these parameters has a natural interpretation for linear codes, and although the six basic modification techniques are not restricted to linear codes it will be easy initially to describe them in these terms. Each fixes one parameter and increases or decreases the other two parameters accordingly. We have:

(*i*) *Augmenting.* Fix n; increase k; decrease r.
(*ii*) *Expurgating.* Fix n; decrease k; increase r.
(*iii*) *Extending.* Fix k; increase n; increase r.
(*iv*) *Puncturing.* Fix k; decrease n; decrease r.
(*v*) *Lengthening.* Fix r; increase n; increase k.
(*vi*) *Shortening.* Fix r; decrease n; decrease k.

The six techniques fall naturally into three pairs, each member of a pair the inverse process to the other. Since the redundancy of a code is its "dual dimension," each technique also has a natural dual technique.

6.1.1 Augmenting and expurgating

In augmenting or expurgating a code we keep its length fixed but vary its dimension and redundancy.

When *augmenting* a code C we add codewords to C. augmenting

expurgating

The inverse process of *expurgating* a code is the throwing out of codewords. Notice that augmentation may cause the minimum distance to decrease, while expurgation will not decrease minimum distance and may, in fact, increase it. For generalized Reed-Solomon codes, we always have

$$GRS_{n,k-1}(\boldsymbol{\alpha}, \mathbf{v}) \leq GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v}) .$$

Therefore the second code results from augmenting the first, and the first from expurgating the second. In this case the expurgated code has larger minimum distance.

A linear code can be easily augmented by adding rows to a generator matrix and expurgated by taking away rows. A typical way of augmenting a linear code is by adding the row vector composed entirely of 1's to its generator matrix. (Of course, in certain cases this vector will already belong to the code; so the action is inappropriate.)

Increasing the size of a linear code is the same as decreasing the size of its dual, so these two techniques are dual to each other as well as inverse. The dual of augmenting a code by the all 1's vector is expurgating by keeping only those codewords whose entries sum to 0.

These techniques are used for nonlinear codes as well. Consider a linear code C that is a subcode of the linear code D. We can create new codes, not necessarily linear, that are augmentations of C and expurgations of D, by taking the union of certain cosets of C in D. For instance, we might choose those cosets whose coset leaders had largest weight. This method has produced some nonlinear codes that have better properties (in this case minimum distance) than any linear code with the same length and size.

If K is a subfield of the field F and C is a code over F, then we can expurgate C by keeping only those codewords all of whose entries belong to K. This

subfield subcode

subfield subcode inherits many of the properties of the original code and may have further nice properties as well. This extremely important type of expurgation will be discussed at length in a later chapter.

6.1.2 Extending and puncturing

In extending or puncturing a code we keep its dimension fixed but vary its length and redundancy. These techniques are exceptional in that they are one-to-one. Issues related to the extending and puncturing of GRS codes will be discussed in the next two sections.

extending

puncturing

When *extending* a code we add extra redundancy symbols to it. The inverse is *puncturing*, in which we delete redundancy symbols. Puncturing may cause the minimum distance to decrease, but extending will not decrease the minimum distance and may, in fact, increase it. (See Problem 6.1.1 below.) To extend a linear code we add columns to its generator matrix, and to puncture the code we delete columns from its generator.

coordinate extension

Let us call the $[n+1, k]$ linear code C^+ a *coordinate extension* of C if it results from the addition of a single new redundancy symbol to the $[n, k]$ linear

code C over the field F. Each codeword $\mathbf{c}^+ = (c_1, \ldots, c_n, c_{n+1})$ of the extended code C^+ is constructed by adding to the codeword $\mathbf{c} = (c_1, \ldots, c_n)$ of C a new coordinate $c_{n+1} = \sum_{i=1}^{n} a_i c_i = \mathbf{a} \cdot \mathbf{c}$, for some fixed $\mathbf{a} = (a_1, \ldots, a_n) \in F^n$. Here we imply that the new coordinate is the last one, but this is not necessary. A coordinate extension can add a new position at any place within the original code.

Although it is formally possible, it makes little sense to add a coordinate determined by a vector $\mathbf{a}$ of $C^\perp$, since each new c_{n+1} would be 0. We thus assume that $\mathbf{a} \notin C^\perp$. In this case, the subcode $C_0 = C \cap \mathbf{a}^\perp$ of C has dimension $k-1$. We call C_0 the *kernel* of the extension. Replacing the vector $\mathbf{a}$ by any other vector of the coset $\mathbf{a} + C^\perp$ leaves the extension C^+ and the kernel C_0 unchanged. Replacing $\mathbf{a}$ by a nonzero scalar multiple gives an extension of C diagonally equivalent to C^+ and with the same kernel C_0.

kernel

The kernel C_0 is that subcode of C that has 0 added in the extension position c_{n+1} of C^+. If G_0 is a generator matrix for the kernel C_0 and $\mathbf{c} \in C - C_0$, then one generator matrix for the coordinate extension C^+ is

$$\left[\begin{array}{c|c} G_0 & \mathbf{0} \\ \hline \mathbf{c} & c_{n+1} \end{array}\right],$$

where $\mathbf{0}$ is a column vector of $k-1$ entries equal to 0. Conversely, for any linear $[n, k-1]$ subcode C_0 of C, there is a coordinate extension C^+ of C with kernel C_0. The extension C^+ can be constructed via a generator matrix as above or, equivalently, by choosing a vector $\mathbf{a}$ in $C_0^\perp - C^\perp$.

The most typical method of extending a code is the appending of an overall parity check symbol, a final symbol chosen so that the entries of each new codeword sum to zero. This corresponds to a coordinate extension in which $\mathbf{a}$ has all of its entries equal to -1. For binary codes, this is the usual requirement that the 1's of a codeword in the extended code have even parity. For C a binary Hamming code, this leads to the extended Hamming codes as constructed in Section 4.3, although there we added the new coordinate at the front of the codeword rather than the rear. An extended binary Hamming code has minimum distance 4. No matter what position we choose to puncture an extended binary Hamming code, we are left with a code of minimum distance 3. This code must again be a Hamming code by Problem 4.1.3.

(6.1.1) PROBLEM. *Let C^+ be a coordinate extension of C with kernel C_0. If $d_{min}(C_0) > d_{min}(C)$, prove that $d_{min}(C^+) = d_{min}(C) + 1$.*

Let $\mathbf{x}'$ denote the vector of length $n-1$ that is gotten by deleting the last entry from the vector $\mathbf{x}$ of length n. Then $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ can be punctured to $GRS_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. Clearly this can be repeated and not always in the final coordinate. The relationship between puncturing and correction of erasures in GRS codes is discussed in the next section of this chapter. On the other hand $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is an extension of $GRS_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. Extensions of this kind are always possible as long as $n < |F|$. We talk about further extension of GRS codes in the final section of the chapter.

(6.1.2) PROBLEM. *Let C be a code with minimum distance d. Prove that C can correct any pattern of g erasures and e errors provided*

$$g + 2e + 1 \leq d\,.$$

*(*HINT: *For a given pattern, consider the code that is C punctured at the erasure locations.)*

6.1.3 Lengthening and shortening

In lengthening or shortening a code we keep its redundancy fixed but vary its length and dimension.

lengthening
shortening

When *lengthening* a code C we increase the length and add codewords to C. The inverse process of *shortening* a code involves the throwing out of codewords and deleting coordinate positions. As such, these operations can be thought of as combinations of the ones discussed above. Lengthening is extending followed by augmenting, and shortening is expurgating followed by puncturing. Since the two constituent operations tend to have opposing influence on the minimum distance, the actual effect of a lengthening or shortening operation upon distance will depended upon the situation.

For linear codes lengthening corresponds to bordering a generator matrix by adding new columns (extending) and the same number of new rows (augmenting). A standard method is to add to the original generator a final column that is entirely 0, and then add a row that is nonzero in this new column, for instance, the vector of all 1's. Thus a coordinate extension D^+ of a linear code D is a lengthening of its kernel $C = D_0$. Lengthening a code is dual to extending, and the special case of adding an all 0 column and all 1 row for C corresponds to extending $C^\perp$ by an overall parity check symbol. Thus in Section 4.3, we started with a lexicographic generator matrix L_m for the dual Hamming code C and bordered it to construct a generator EL_m for the first order Reed-Muller code $RM(1, m)$ whose dual is the extended Hamming code.

(6.1.3) PROBLEM. *Let C^+ be a coordinate extension of the linear code C with kernel C_0. Prove that $(C^+)^\perp$ is an extension of $C_0^\perp$ and a lengthening of $C^\perp$.*

Shortening undoes lengthening by removing a border from a generator matrix. To reverse the standard 0 column lengthening just described, we first find a generator matrix for the longer code that has a unique row in which the last column is nonzero. Then delete that row (expurgating) and the final column (puncturing), leaving a generator matrix for the original code. In fact this reconstructs the original code as the kernel of a coordinate extension in the overall parity check position. Of course this type of shortening can be done with respect to any column. There will also be various other shortenings available, corresponding to deleting borders whose columns have more nonzero entries. Shortening plays a role in constructing noncyclic CRC codes from cyclic codes, as discussed in Section 8.4.1.

Using the $\mathbf{x}$, $\mathbf{x}'$ notation of the previous subsection, we see that the canonical generator matrix for the code $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is obtained by bordering the

canonical generator matrix for $GRS_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$. Therefore we can shorten $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ to $GRS_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$. In general this will not be the same as the $[n-1, k-1]$ shortened code constructed above by deleting the last position from all codewords that finish with 0, the kernel of $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ viewed as an extension in its last coordinate.

(6.1.4) PROBLEM. *Let C be $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$, and shorten C to the $[n-1, k-1]$ code D by taking all codewords of C that end in 0 and then deleting this last coordinate. Find vectors $\boldsymbol{\beta}$ and $\mathbf{u}$ with $D = GRS_{n-1,k-1}(\boldsymbol{\beta}, \mathbf{u})$.*

We can also lengthen and shorten nonlinear codes. Choose a coordinate position, and select from the code only those words that have some fixed entry in that place. Then delete that position from each of these words. This process can be repeated any number of times, leaving the residue of all codewords that match some specific pattern on some specific set of positions, those positions then deleted. We can use this approach to prove the rest of the Asymptotic Plotkin Bound 2.3.9(2):

$$\alpha_m(\delta) \leq 1 - \frac{m}{m-1}\delta\,, \quad \text{for} \quad 0 \leq \delta \leq \frac{m-1}{m}\,.$$

PROOF OF COROLLARY 2.3.9.

Consider a family $\{C_n\}$ of m-ary codes of unbounded length n and such that the limits

$$\lim_{n\to\infty} d(C_n)/n = \lim_{n\to\infty} \delta(C_n) = \delta$$

and

$$\lim_{n\to\infty} k(C_n)/n = \lim_{n\to\infty} \kappa(C_n) = \kappa$$

both exist. Assume additionally that $\delta \leq (m-1)/m$. We wish to prove

$$\kappa \leq 1 - \frac{m}{m-1}\delta\,.$$

Clearly we may assume that $\delta \neq 0$, so $d(C_n)$ goes to infinity with n. In particular, there is an integer N such that $d(C_n) \geq m$, for all $n > N$.

Let $n > N$, and set $C = C_n$ and $d = d(C_n)$. Define

$$n' = \left\lfloor (d-1)\frac{m}{m-1} \right\rfloor = (d-1) + \left\lfloor \frac{d-1}{m-1} \right\rfloor \geq d\,.$$

Since $1 > (m-1)/m \geq 0$,

$$d - 2 < \frac{m-1}{m} n' \leq d - 1\,.$$

Let $\mathbf{x}$ be an $(n-n')$-tuple. Shorten the code to a set $C' = C'(\mathbf{x})$ of n'-tuples by choosing all codewords ending in the $(n-n')$-tuple $\mathbf{x}$ and then deleting these

last $(n - n')$ positions from the chosen words. Then either C' is empty or the shortened code C' has length n' and minimum distance $d' \geq d$. Furthermore

$$\frac{m-1}{m} < \frac{d}{n'} \leq \frac{d'}{n'}\,,$$

Therefore the Plotkin Bound 2.3.8 can be applied to C' to yield

$$|C'| \leq \frac{d'}{d' - \frac{m-1}{m}n'} \leq \frac{d}{d - \frac{m-1}{m}n'} \leq d\,,$$

since the function $f(x) = x/(x-c)$ is decreasing. There are $m^{n-n'}$ possible choices for $\mathbf{x}$, and each $C'(\mathbf{x})$ has size at most d; so

$$|C| \leq dm^{n-n'}\,.$$

Taking logarithms and dividing by n, we reach

$$\begin{aligned} \frac{\log_m(|C|)}{n} &\leq \frac{\log_m(d) + n - n'}{n} \\ &\leq \frac{\log_m(n)}{n} + 1 - \frac{m}{m-1}\frac{d-2}{n}\,. \end{aligned}$$

Therefore, for all $n > N$,

$$\kappa(C_n) \leq 1 - \frac{m}{m-1}\delta(C_n) + n^{-1}\left(\log_m(n) + 2\frac{m}{m-1}\right)\,,$$

which, in the limit, is the desired bound. □

6.2 Puncturing and erasures

In Subsection 6.1.2 we saw that the correction of erasures and errors can be dealt with through correction of errors for a suitable punctured code. Indeed the following theorem is a special case of Problem 6.1.2.

(6.2.1) THEOREM. *The code $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ can be used to correct any pattern of g erasures and e errors provided*

$$g + 2e \leq n - k\,.$$

We are interested in proving the theorem by displaying a specific algorithm for decoding. We shall see that a simple modification of Euclidean algorithm decoding allows us to find the error and erasure locations, at which point the algorithm of Proposition 3.3.3 can be used to find all values.

Remember that for the code $C = GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ we defined

$$L(x) = \prod_{i=1}^{n}(x - \alpha_i)$$

and

$$L_i(x) = L(x)/(x - \alpha_i)\,.$$

Let J be a subset of the coordinate positions, and consider the code $C_J = GRS_{n-g,k}(\boldsymbol{\alpha}_J, \mathbf{v}_J)$, gotten by puncturing C at the coordinate positions of the set $\bar{J}$, the complement of J, with $|\bar{J}| = g$. For C_J we have

$$L_J(x) = \prod_{i \in J}(x - \alpha_i)$$

and

$$L_{J,i}(x) = L_J(x)/(x - \alpha_i)\,,$$

for $i \in J$. If we let

$$M_J(x) = \prod_{i \notin J}(x - \alpha_i) = \prod_{i \in \bar{J}}(x - \alpha_i)\,,$$

then

$$L(x) = L_J(x)M_J(x) \text{ and } L_i(x) = L_{J,i}(x)M_J(x)\,.$$

By Theorem 5.1.6 the dual of C is $GRS_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u})$, where

$$u_i = \frac{1}{v_i L_i(\alpha_i)}\,;$$

and the dual of C_J is $GRS_{n-g,n-g-k}(\boldsymbol{\alpha}_J, \tilde{\mathbf{u}})$, where

$$\tilde{u}_i = \frac{1}{v_i L_{J,i}(\alpha_i)}\,,$$

for $i \in J$. Notice that we do not get $\tilde{\mathbf{u}}$ by simple puncturing of $\mathbf{u}$ (and so we do not write $\mathbf{u}_J$.) Nevertheless $\tilde{\mathbf{u}}$ is easy to calculate. For $i \in J$,

$$\begin{aligned}
\tilde{u}_i &= \frac{1}{v_i L_{J,i}(\alpha_i)} \\
&= \frac{1}{v_i (L_i(\alpha_i)/M_J(\alpha_i))} \\
&= \frac{M_J(\alpha_i)}{v_i L_i(\alpha_i)} \\
&= M_J(\alpha_i) u_i\,.
\end{aligned}$$

We have proven

(6.2.2) PROPOSITION. *The dual of $GRS_{n-g,k}(\boldsymbol{\alpha}_J, \mathbf{v}_J)$ is*

$$GRS_{n-g,n-g-k}(\boldsymbol{\alpha}_J, \tilde{\mathbf{u}})\,,$$

where $\tilde{u}_i = M_J(\alpha_i)u_i$. □

We return to Theorem 6.2.1. Suppose we receive the word $\mathbf{p}$ which contains g erasures at the positions $\bar{J}$. To find the locations of the errors (as opposed to erasures) we decode the punctured received word $\mathbf{p}_J$ using the punctured code C_J. As C_J corrects $\lfloor (n-g-k+1)/2 \rfloor$ errors, this already proves the theorem without an algorithm.

We now describe the error location algorithm, following that of Theorem 5.2.3. We first calculate the J-syndrome:

$$\begin{aligned} S_J(z) &= \sum_{i \in J} \frac{M_J(\alpha_i) u_i p_i}{1 - \alpha_i z} \pmod{z^{r-g}} \\ &= \sum_{i=1}^{n} \frac{M_J(\alpha_i) u_i p_i}{1 - \alpha_i z} \pmod{z^{r-g}} . \end{aligned}$$

Next we step through the Euclidean algorithm with the initialization

$$a(z) = z^{r-g} \text{ and } b(z) = S_J(z)$$

until a step j is reached where $\deg(r_j(z)) < (r-g)/2$. We can then find the error locations I in J from the calculated $\sigma_J(z)$ (and $\omega_J(z)$).

Of course we could at the same time find the error values at the locations in I, but we would still need to find the values at the erasure locations in $\bar{J}$. It is probably more efficient to use the algorithm of Proposition 3.3.3 to find all values at the $g + e \leq d - 1$ error and erasure locations $I \cup \bar{J}$ simultaneously.

6.3 Extended generalized Reed-Solomon codes

Let $n > 1$, and consider n-tuples from the field F with the following properties:
(i) $\mathbf{w} = (w_1, w_2, \ldots, w_n) \in F^n$ has all its entries w_i not 0;
(ii) $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n) \in F^n$ and $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_n) \in F^n$ satisfy

$$\beta_i \gamma_j \neq \beta_j \gamma_i , \text{ for all } i \neq j .$$

extended generalized Reed-Solomon code

For $k > 0$ the *extended generalized Reed-Solomon code* $EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ is the code C composed of all codewords

$$\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f) = (w_1 f(\beta_1, \gamma_1), \ldots, w_i f(\beta_i, \gamma_i), \ldots, w_n f(\beta_n, \gamma_n)) ,$$

where $f = f(x, y)$ runs through all polynomials of $F[x, y]$ that are homogeneous of degree $k - 1$:

$$f(x, y) = f_0 y^{k-1} + f_1 x y^{k-2} + f_2 x^2 y^{k-3} + \cdots + f_{k-1} x^{k-1} \text{ with } f_i \in F .$$

The condition (ii) states that, for all distinct i, j, there is no $c \in F$ with $(\beta_i, \gamma_i) = c(\beta_j, \gamma_j)$. It should be thought of as saying that

$$\beta_i / \gamma_i \neq \beta_j / \gamma_j , \text{ for all } i \neq j ,$$

but care must be taken since we allow the possibility $\gamma_j = 0$. There is at most one j with $\gamma_j = 0$, and in that case $\beta_j \neq 0$ since $n > 1$.

For each i, let α_i be the ratio β_i/γ_i, where we write ∞ for $\beta_j/\gamma_j = \beta_j/0$. Then by (ii) the α_i are all distinct. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2 \ldots, \alpha_n)$. Further let $\boldsymbol{\alpha}'$ be $\boldsymbol{\alpha}$ punctured at position j where $\alpha_j = \infty$, if such a position exists.

For each i, set $v_i = w_i \gamma_i^{k-1}$; and let $\mathbf{v} = (v_1, v_2, \ldots, v_n)$. All entries of $\mathbf{v}$ are nonzero with the possible exception of that v_j where $\gamma_j = 0$ and $\alpha_j = \infty$, if such a j exists. In that case let $\mathbf{v}'$ be $\mathbf{v}$ punctured at position j.

We first check that the code we have defined is really an extension of a generalized Reed-Solomon code.

(6.3.1) THEOREM. *Let $C = EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$. If $\alpha_j = \infty$ and $\gamma_j = 0$ then the code gotten by puncturing C at position j is $GRS_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. If no such j exists, then $C = GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$.*

PROOF. Let C' be the code gotten by puncturing C at j where $\alpha_j = \infty$. If no such j exists, let $C' = C$.

With each degree $k-1$ homogeneous polynomial $f(x, y)$ as above, we associate a polynomial $\hat{f}$ in the single indeterminate $\frac{x}{y}$:

$$\hat{f}\left(\frac{x}{y}\right) = \frac{1}{y^{k-1}} f(x, y) = f_0 + f_1 \left(\frac{x}{y}\right) + f_2 \left(\frac{x}{y}\right)^2 + \cdots + f_{k-1} \left(\frac{x}{y}\right)^{k-1} .$$

The polynomial $\hat{f}(\frac{x}{y})$ has degree at most $k-1$ and satisfies

$$\hat{f}(\alpha_i) = \frac{1}{\gamma_i^{k-1}} f(\beta_i, \gamma_i) .$$

Therefore, for any i with $\alpha_i \neq \infty$, the i^{th} entry of the codeword

$$\mathbf{ev}_{\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w}}(f)$$

in the code $C = EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ equals that of the codeword

$$\mathbf{ev}_{\boldsymbol{\alpha}', \mathbf{v}}(\hat{f})$$

in the generalized Reed-Solomon code $GRS_{n',k}(\boldsymbol{\alpha}', \mathbf{v}')$. That is,

$$C' = GRS_{n',k}(\boldsymbol{\alpha}', \mathbf{v}') . \qquad \square$$

A canonical generator matrix for C has rows $\mathbf{ev}_{\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w}}(f)$ as $f = f(x, y)$ runs through the basis $y^i x^{k-1-i}$ of the space of homogeneous polynomials of degree $k-1$. This matrix is also obtained by adding to the canonical generator matrix for $GRS_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$ at position j (where $\alpha_j = \infty$) a column that is all 0 except for the entry $w_j \beta_j^{k-1}$ in its last row. (Compare Problem 5.1.4.) In particular $GRS_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$ is revealed as the kernel of the coordinate

extension of $GRS_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$ at position j that produces the extended code $EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$. In particular, the next theorem is a consequence of the previous theorem and Problem 6.1.1. Instead we give a proof following that of the corresponding result for *GRS* codes, Theorem 5.1.1.

(6.3.2) THEOREM. *The code $EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ is an $[n, k]$ linear code over F with minimum distance $n - k + 1$.*

PROOF. The only thing that needs careful checking is that the minimum distance is at least $n - k + 1 = n - (k - 1)$.

Let $f(x, y)$ be a homogeneous polynomial of degree $k - 1$, and let $\hat{f}(\frac{x}{y})$ be its associated polynomial. As all the w_i are nonzero, the number of entries 0 in $\mathbf{f} = \mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f)$ equals the number of i with $f(\beta_i, \gamma_i) = 0$. We must prove there are at most $k-1$ such i. There are two cases to consider, depending upon whether or not $f(\beta_j, \gamma_j) = 0$ for a j with $\gamma_j = 0$ and $\alpha_j = \infty$.

First consider those 0's of $\mathbf{f}$ that occur at positions i for which $\gamma_i \neq 0$. Each corresponding α_i is a root of the polynomial $\hat{f}$, and there are at most $\deg(\hat{f})$ roots. In particular, in the case where all 0's of $\mathbf{f}$ occur at such positions i, there are at most $k - 1 \leq \deg(\hat{f})$ places equal to 0, as required.

Now assume that $\gamma_j = 0$ and $f(\beta_j, 0) = 0$, that is,

$$\begin{aligned} 0 &= f_0 0^{k-1} + f_1 \beta_j 0^{k-2} + f_2 \beta_j^2 0^{k-3} + \cdots + f_{k-2} \beta_j^{k-2} 0^1 + f_{k-1} \beta_j^{k-1} \\ &= f_{k-1} \beta_j^{k-1} . \end{aligned}$$

As $\beta_j \neq 0$, we must have $f_{k-1} = 0$ in this case. Therefore the degree of $\hat{f}$ is in fact at most $k-2$. So even here there are at most $k-1$ places where $\mathbf{f}$ is 0, one at position j and at most $\deg(\hat{f}) \leq k - 2$ at other locations. □

(6.3.3) PROBLEM. *Prove that the dual of an EGRS code is also an EGRS code.*

(6.3.4) THEOREM. *Let $a, b, c, d, e \in F$ with*

$$ad - bc \neq 0 \text{ and } e \neq 0 .$$

Then

$$EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w}) = EGRS_{n,k}(\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\gamma}}; \tilde{\mathbf{w}}) ,$$

where

$$\begin{aligned} \tilde{\beta}_i &= a\beta_i + b\gamma_i, \\ \tilde{\gamma}_i &= c\beta_i + d\gamma_i, \\ \text{and } \tilde{w}_i &= e w_i . \end{aligned}$$

PROOF. The proof consists mainly of calculation. The crucial observation is that, for any homogeneous polynomial $f(x, y)$ of degree $k - 1$ and for the quadruple $r, s, t, u \in F$, the polynomial $f(rx + sy, tx + uy)$ is also homogeneous of degree $k - 1$, provided $rx + sy \neq 0 \neq tx + uy$.

The number $\Delta = ad - bc$ is the determinant of the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} .$$

As Δ is nonzero we can solve for β_i and γ_i and find that

$$\begin{aligned} \beta_i &= \Delta^{-1}(d\tilde{\beta}_i - b\tilde{\gamma}_i) , \\ \gamma_i &= \Delta^{-1}(-c\tilde{\beta}_i + a\tilde{\gamma}_i) . \end{aligned}$$

As $e \neq 0$, the vector $\tilde{\mathbf{w}}$ has no entries 0, since $\mathbf{w}$ has none. We also check that

$$\tilde{\beta}_i\tilde{\gamma}_j - \tilde{\beta}_j\tilde{\gamma}_i = \Delta(\beta_i\gamma_j - \beta_j\gamma_i) \neq 0 ,$$

giving the defining conditions (i) and (ii) for the vectors $\tilde{\boldsymbol{\beta}}$, $\tilde{\boldsymbol{\gamma}}$, and $\tilde{\mathbf{w}}$.

Starting with the degree $k-1$ homogeneous polynomial $f(x, y)$, we define the new polynomial

$$g(x, y) = \frac{1}{e\Delta^{k-1}} f(dx - by, -cx + ay) .$$

Then

$$\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f) = \mathbf{ev}_{\tilde{\boldsymbol{\beta}},\tilde{\boldsymbol{\gamma}};\tilde{\mathbf{w}}}(g) .$$

Therefore each codeword of the first code is also in the second code. As both codes have the same dimension, they must be equal. □

Problems 5.1.2 and 5.1.3 are special cases of this theorem.

The $q+1$ possible ratios $\alpha_i = \beta_i/\gamma_i$ from $\{\infty\} \cup \mathbb{F}_q$ are identified with the projective line over $\mathbb{F}_q$. The *EGRS* codes can thus be thought of as codes defined by functions on the projective line. The group of 2×2 matrices that appears in Theorem 6.3.4 acts naturally on the projective line.

(6.3.5) THEOREM. *If $n \leq |F|$, then $C = EGRS_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ is equal to $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ over F, for appropriate $\boldsymbol{\alpha}$ and $\mathbf{v}$. If $n < |F|$, then $\boldsymbol{\alpha}$ may be chosen with all its entries not equal to 0.*

PROOF. If $n \leq |F|$, then some possible ratio α does not occur among the $\alpha_i = \beta_i/\gamma_i$. If the ratio $\alpha = \infty$ is missing, then C is a *GRS* code by Theorem 6.3.1. If $\gamma_j = 0$ and $\alpha \neq \infty$, then any transformation

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ -1 & \alpha \end{bmatrix}$$

in Theorem 6.3.4 takes $\boldsymbol{\gamma}$ to a vector $\tilde{\boldsymbol{\gamma}}$ with no entry 0; so C is again a *GRS* code by Theorem 6.3.1. If $n < |F|$ then the the values of a, b, c, and d in the transformation can be chosen so that both $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$ avoid 0. Then $C = GRS_{n,k}(\tilde{\boldsymbol{\alpha}}, \mathbf{v})$ with each entry $\tilde{\alpha} = \tilde{\beta}/\tilde{\gamma}$ of $\tilde{\boldsymbol{\alpha}}$ nonzero. □

Chapter 7

Codes over Subfields

In Chapter 6 we looked at various general methods for constructing new codes from old codes. Here we concentrate on two more specialized techniques that result from writing the field F as a vector space over its subfield K. We will start with linear codes over F and finish with linear codes over K. Of particular practical importance is the case with $K = \mathbb{F}_2$. Our work on generalized Reed-Solomon codes over F has given us many powerful codes, but by Theorem 5.1.1 their length is bounded by $|F|$. Binary generalized Reed-Solomon codes are rendered trivial.

7.1 Basics

Let $\dim_K(F) = m$, and choose $e_1, \ldots e_m$ to be a K-basis for F. We define the map $\phi\colon F \longrightarrow K^m$ given by

$$\phi(\alpha) = (a_1, \ldots a_m) \text{ where } \alpha = a_1 e_1 + \cdots + a_m e_m \,.$$

For brevity, we shall write $\hat{\alpha}$ for the $1 \times m$ row vector $\phi(\alpha)$ and $\check{\alpha}$ for its transpose $\phi(\alpha)^\top = (a_1, \ldots a_m)^\top$, an $m \times 1$ column vector. We extend this notation to any $p \times q$ matrix $A \in F^{p,q}$, with i, j entry $a_{i,j}$ by letting $\widehat{A} \in K^{p,mq}$ be the matrix

$$\begin{bmatrix} \hat{a}_{1,1} & \hat{a}_{1,2} & \cdots & \hat{a}_{1,j} & \cdots & \hat{a}_{1,q} \\ \hat{a}_{2,1} & \hat{a}_{2,2} & \cdots & \hat{a}_{2,j} & \cdots & \hat{a}_{1,q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{a}_{i,1} & \hat{a}_{i,2} & \cdots & \hat{a}_{i,j} & \cdots & \hat{a}_{1,q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{a}_{p,1} & \hat{a}_{p,2} & \cdots & \hat{a}_{p,j} & \cdots & \hat{a}_{p,q} \end{bmatrix}$$

and $\check{A} \in K^{mp,q}$ be the matrix

$$\begin{bmatrix} \check{a}_{1,1} & \check{a}_{1,2} & \cdots & \check{a}_{1,j} & \cdots & \check{a}_{1,q} \\ \check{a}_{2,1} & \check{a}_{2,2} & \cdots & \check{a}_{2,j} & \cdots & \check{a}_{1,q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \check{a}_{i,1} & \check{a}_{i,2} & \cdots & \check{a}_{i,j} & \cdots & \check{a}_{1,q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \check{a}_{p,1} & \check{a}_{p,2} & \cdots & \check{a}_{p,j} & \cdots & \check{a}_{p,q} \end{bmatrix}$$

For our constructions, A might be a spanning or control matrix for a linear code over F. Then the matrices $\widehat{A}$ and $\check{A}$ can be thought of as spanning or control matrices for linear codes over K.

It must be emphasized that these maps are highly dependent upon the choice of the initial map ϕ, even though ϕ has been suppressed in the notation. We shall see below that a careful choice of ϕ can be of great help. (The general situation in which ϕ is an arbitrary injection of F into K^p, for some field K and some p, is of further interest. Here we will be concerned with the linear case, but see the Problem 7.2.3 below.)

7.2 Expanded codes

If C is a code in F^n, then the code

$$\widehat{C} = \{\, \hat{\mathbf{c}} \,|\, \mathbf{c} \in C \,\}$$

expanded code

in K^{mn} is called an *expanded code*.

(7.2.1) THEOREM. *If C is an $[n, k, d]$ code, then $\widehat{C}$ is an $[mn, mk, \geq d]$ code.*

PROOF. The map $\mathbf{x} \mapsto \hat{\mathbf{x}}$ (induced by ϕ) is one-to-one and has

$$\widehat{r\mathbf{a} + s\mathbf{b}} = r\hat{\mathbf{a}} + s\hat{\mathbf{b}}\,,$$

for all $r, s \in K$ and $\mathbf{a}, \mathbf{b} \in F^n$. Thus $\widehat{C}$ is a linear code over K with

$$|\widehat{C}| = |C| = |F|^k = (|K|^m)^k = |K|^{mk}\,,$$

hence $\widehat{C}$ has K-dimension mk. (This counting argument is a cheat unless F is finite, the case of greatest interest to us. Instead, one should construct a K-basis for $\widehat{C}$ out of that for F and an F-basis for C. We do this below, constructing a generator matrix for $\widehat{C}$ from one for C.)

If the coordinate c_i of $\mathbf{c}$ is nonzero, then $\hat{c}_i$ is not the zero vector of K^m. Therefore each nonzero entry in $\mathbf{c}$ corresponds to a nonzero m-tuple $\hat{c}_i$ within $\hat{\mathbf{c}}$ and $w_H(\mathbf{c}) \leq w_H(\hat{\mathbf{c}})$. In particular $d_{min}(C) \leq d_{min}(\widehat{C})$. □

The argument of the last paragraph shows that we would be very unlucky indeed to have $d_{min}(C) = d_{min}(\widehat{C})$. For this to happen we would need a minimum weight codeword $\mathbf{c}$ in C for which every nonzero c_i had $\hat{c}_i$ of weight 1. In

the next section we shall see two examples in which the minimum distance of an expanded codes goes up over that of its parent.

Let G be a generator matrix for C with rows $\mathbf{g}_i$, for $i = 1, \ldots, k$. Notice that $\mathbf{g}_i$ and $e_j\mathbf{g}_i$ are F-scalar multiples but are linearly independent over K. The mk vectors $e_j\mathbf{g}_i$ (for $1 \le j \le m$, $1 \le i \le k$) form a basis for C, thought of as a K-space of dimension mk. If we let G_0 be the $mk \times n$ matrix whose rows are the various $e_j\mathbf{g_i}$, then G_0 is a spanning matrix for C and $\widehat{G}_0$ is a generator matrix for $\widehat{C}$.

A vector is a *burst* of length f if all of its nonzero entries are restricted to a set of f consecutive positions. For instance, (00011101000) is a burst of length 5 (and 6, 7, ..., 11, as well). Certain channels are prone to burst errors. (Think of a scratch on a CD.) Expanded codes give some easy protection against burst errors, since an error in m consecutive positions of $\widehat{C}$ corresponds to only one or two errors for the parent code C. burst

(7.2.2) PROPOSITION. *If the linear code C can be used to correct burst errors of length e (in particular, if C is an e error-correcting code), then $\widehat{C}$ can be used to correct all burst errors of length up to $1 + (e-1)m$.*

PROOF. A burst in K^{mn} of length at most $1 + (e-1)m$ has as preimage in F^n a burst of length at most e. □

(7.2.3) PROBLEM. (*a*) *Consider an injection ϕ of $\mathbb{F}_4$ into $\mathbb{F}_2^3$ with the property that*

$$\phi(\mathbb{F}_4) = \{001,\ 110,\ 010,\ 101\}\,.$$

Prove that, for any code $C \subseteq \mathbb{F}_4^n$, the corresponding expanded code $\hat{C}$ in $\mathbb{F}_2^{3n}$ has the property that each codeword has no more than three consecutive 0's and no more than three consecutive 1's among its entries. (This is a 'run-length-limited' constraint of the sort that is made for magnetic recording and on compact discs.)

(*b*) *Prove that there are exactly four 4-subsets of $\mathbb{F}_2^3$ with the property discussed in* (*a*).

Expanding is often used as an easy way to construct good binary codes for bursty channels from codes over larger fields of characteristic 2. For instance, one code that has been used by NASA and the European Space Agency, (for space communication) and IBM, Phillips, and Sony (for tape and CD storage) is the binary expansion of a *GRS* code of length $255\,(= 2^8 - 1)$ and dimension 223 over $\mathbb{F}_{2^8}$. (The vector $\boldsymbol{\alpha}$ contains all nonzero field elements as entries, while $\mathbf{v} = \mathbf{1}$.) Expanding from $\mathbb{F}_{2^8}$ to $\mathbb{F}_2$ (so that $m = 8$) allows symbols of the *GRS* code to be written as bytes of data. The associated binary expanded code has length $mn = 8(255) = 2040$ and dimension $mk = 8(223) = 1784$. The parent *GRS* code has $d_{min} = 255 - 223 + 1 = 33$, so it can correct up to 16 errors. Therefore the stretched code can correct any burst error of length at most $1 + (16-1)8 = 121$ as well as any random error of weight at most 8.

7.3 Golay codes and perfect codes

We construct four of the most famous and important codes using expanded codes.

7.3.1 Ternary Golay codes

Here we have $F = \mathbb{F}_9$, $K = \mathbb{F}_3$, and $m = 2$ in Theorem 7.2.1.

Let i be a root of the imprimitive polynomial $x^2 + 1 \in \mathbb{F}_3[x]$. We then write the field $\mathbb{F}_9$ as $\{a + bi \,|\, a, b \in \mathbb{F}_3\}$, having chosen the $\mathbb{F}_3$-basis of $e_1 = 1$ and $e_2 = i$ for $\mathbb{F}_9$, so that the associated expansion map is

$$\beta = a\,1 + b\,i \mapsto \phi(\beta) = \hat{\beta} = (a, b)\,,$$

for $a, b \in \mathbb{F}_3$. For each $\beta = a + bi \in \mathbb{F}_9$, let $\bar{\beta} = a - bi$, the conjugate of β.

Let A be a unitary 3×3 matrix with entries from $\mathbb{F}_9$, that is $A\bar{A}^\top = I$; and let $\alpha \in \mathbb{F}_9$ satisfy $\alpha\bar{\alpha} = -1$. Here by $\bar{A}$ we mean the matrix whose i, j entry is $\bar{a}_{i,j}$, where $a_{i,j}$ is the i, j entry of A.

EXAMPLE.

$$A = \begin{bmatrix} 1+i & i & i \\ i & 1+i & i \\ i & i & 1+i \end{bmatrix} \text{ and } \alpha = 1 - i\,.$$

Consider then the $[6, 3]$ linear code C over $\mathbb{F}_9$ with generator matrix

$$G = [\, I \;;\; \alpha A \,]\ ,$$

for example,

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

We then may calculate

$$G\bar{G}^\top = I + \alpha\bar{\alpha}A\bar{A}^\top = I + (-1)I = 0\,;$$

so

$$H = \bar{G} = \left[\, I \;;\; \bar{\alpha}\bar{A} \,\right]$$

is a check matrix for C. In particular $C^\perp$ equals $\bar{C}$, the code composed of the various $\bar{\mathbf{c}}$ as $\mathbf{c}$ runs through C. As G has standard form, a second check matrix for C is

$$H' = \left[\, -\alpha A^\top \;;\; I \,\right]\ .$$

Therefore a second generator matrix is

$$\bar{H}' = \left[\, -\bar{\alpha}\bar{A}^\top \;;\; I \,\right]\ .$$

(7.3.1) PROPOSITION. *Assume that A has no entry equal to 0. Then C has minimum distance 4 and so is an MDS code.*

PROOF. We have $4 = 6 - 3 + 1$, so we must show that C has no codewords of weight 1, 2, or 3. Consider a nonzero codeword $\mathbf{c} = (c^{(1)}; c^{(2)})$ of weight 1, 2, or 3, where $c^{(1)}, c^{(2)} \in \mathbb{F}_9^3$. By the pigeonhole principle, either $c^{(1)}$ or $c^{(2)}$ has weight at most 1.

First suppose $c^{(1)}$ has weight at most 1. In view of the generator matrix G, the only codeword with $c^{(1)}$ equal to 0 is the $\mathbf{0}$-word. A codeword with $c^{(1)}$ of weight 1 is a scalar multiple of some row of G and so has weight 4, since by assumption no entry of A is 0. Thus a nonzero codeword $\mathbf{c}$ with $c^{(1)}$ of weight at most 1 has weight 4.

If instead $c^{(2)}$ has weight at most 1, then we may use the generator matrix $\bar{H}'$ and argue as in the previous paragraph to see again that nonzero $\mathbf{c}$ has weight 4. □

Assume now, as in the example and proposition, that A has been chosen to have none of its entries equal to 0. The $[12, 6, \geq 4]$ ternary code $\widehat{C}$ gotten by expanding C using the map $a + bi \mapsto (a, b)$ is called an *extended ternary Golay code*, as is anything monomially equivalent to it. (For different choices of A this construction will produce different codes $\widehat{C}$, but it turns out that they are all monomially equivalent.)

extended ternary Golay code

If we puncture $\widehat{C}$ at any coordinate position we get an $[11, 6]$ linear code which is called a *ternary Golay code*.

ternary Golay code

(7.3.2) THEOREM. *(*M. GOLAY, 1949.*)* (1) *An extended ternary Golay code is a self-dual* $[12, 6, 6]$ *linear code over* $\mathbb{F}_3$.

(2) *A ternary Golay code is a perfect* 2*-error-correcting* $[11, 6]$ *linear code over* $\mathbb{F}_3$.

PROOF. Let $\mathbf{x} = (x_1, \ldots x_6)$, $\mathbf{y} = (y_1, \ldots y_6) \in \mathbb{F}_9^6$ with $x_j = a_j + b_j i$ and $y_j = c_j + d_j i$. Then we easily find

$$\mathbf{x} \cdot \bar{\mathbf{y}} = \hat{\mathbf{x}} \cdot \hat{\mathbf{y}} + f\, i\,,$$

for some $f \in \mathbb{F}_3$. In particular if $\mathbf{x} \cdot \bar{\mathbf{y}} = 0$, then $\hat{\mathbf{x}} \cdot \hat{\mathbf{y}} = 0$.

Since $C^\perp$ equals $\bar{C}$, the expanded code $\widehat{C}$ is a self-dual ternary $[12, 6]$ linear code. By Problem 3.1.11(b) all weights of $\widehat{C}$ are multiples of 3. By the Singleton Bound 3.1.14 and Theorem 7.2.1

$$12 - 6 + 1 = 7 \geq d_{min}(\widehat{C}) \geq 4 = d_{min}(C)\,,$$

hence $d_{min} = 6$.

Puncturing $\widehat{C}$ at any position, we find a code of minimum distance at least 5; so every ternary Golay code is a 2-error-correcting code. To complete (2) and the theorem, we check equality in the Sphere Packing Condition 2.2.5 for a ternary Golay code:

$$3^{11} \quad \geq \quad 3^6 \sum_{i=0}^{2} \binom{11}{i} (3-1)^i$$

$$\begin{aligned} &= 3^6\Big(\binom{11}{0} + +\binom{11}{1}2 + \binom{11}{2}4\Big) \\ &= 3^6(1+22+220) \\ &= 3^6(243) = 3^6 3^5 = 3^{11}\,. \qquad \square \end{aligned}$$

7.3.2 Binary Golay codes

Here we have $F = \mathbb{F}_8$, $K = \mathbb{F}_2$, and $m = 3$ in Theorem 7.2.1.

Let α be a root in $\mathbb{F}_8$ of the primitive polynomial $x^3 + x + 1 \in \mathbb{F}_2[x]$, and let

$$\boldsymbol{\alpha} = (0, 1, \alpha, \alpha^2, \ldots, \alpha^6) \in \mathbb{F}_8^8\,.$$

In this subsection we begin with the code $D = GRS_{8,4}(\boldsymbol{\alpha}, \mathbf{1})$, which is a self-dual code by Theorem 5.1.6 and Problem 5.1.5(b). As in Problem A.3.18 of the Appendix, choose, as basis for $\mathbb{F}_8$ over $\mathbb{F}_2$, the elements

$$e_1 = \alpha^3,\ e_2 = \alpha^5,\ e_3 = \alpha^6\,.$$

Then, for each $\beta = b_1 e_1 + b_2 e_2 + b_3 e_3$ in $\mathbb{F}_8$, we set

$$\phi(\beta) = \hat{\beta} = (b_1, b_2, b_3)\,.$$

Expand the $[8,4]$ code D over $\mathbb{F}_8 = \mathbb{F}_{2^3}$ to a $[24,12]$ code $\widehat{D}$ over $\mathbb{F}_2$ using this map. Then $\widehat{D}$ or any code equivalent to it is called an *extended binary Golay code*. If we puncture an extended binary Golay code at any coordinate position we get a $[23,12]$ linear code which is called a *binary Golay code*.

extended binary Golay code

binary Golay code

(7.3.3) THEOREM. *(*M. GOLAY, 1949.*)* (1) *An extended binary Golay code is a self-dual* $[24,12,8]$ *linear code over* $\mathbb{F}_2$.

(2) *A binary Golay code is a perfect* 3*-error-correcting* $[23,12]$ *linear code over* $\mathbb{F}_2$.

PROOF. We have already remarked that D is self-dual by Theorem 5.1.6 and Problem 5.1.5(b). Therefore by Theorem 7.2.1 and Problem A.3.18 of the Appendix the extended Golay code $\widehat{D}$ is a self-dual binary $[24,12,\geq 5]$ linear code. As $\widehat{D}$ is self-dual, d_{min} is even by Problem 3.1.11(a) and so at least 6.

Let G be the canonical generator matrix of D with rows $\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3$:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 0 & 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^1 & \alpha^3 & \alpha^5 \\ 0 & 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 \end{bmatrix}\,.$$

As seen earlier, one generator matrix for $\widehat{D}$ has as rows the twelve codewords $\mathbf{c}_{i,j} = \widehat{e_j \mathbf{g}_i}$, for $1 \leq j \leq 3$ and $0 \leq i \leq 3$. Each $\mathbf{c} = \mathbf{c}_{i,j}$ consists of eight binary triples:

$$\mathbf{c} = (c^{(1)}; c^{(2)}; c^{(3)}; c^{(4)}; c^{(5)}; c^{(6)}; c^{(7)}; c^{(8)})\,.$$

If $\mathbf{c} = \mathbf{c}_{0,j}$, then the $c^{(a)}$ are all equal and of weight one, hence $\mathbf{c}$ has weight 8. If $\mathbf{c} = \mathbf{c}_{i,j}$ with $i \neq 0$, then

$$\{\, c^{(a)} \mid 1 \leq a \leq 8 \,\} = \{000, 001, 010, 011, 100, 101, 110, 111\}\,,$$

and $\mathbf{c}$ has weight 12. Therefore in all cases $\mathbf{c} = \mathbf{c}_{i,j}$ has weight a multiple of 4. As these span the self-dual code $\widehat{D}$, Problem 3.1.11(a) guarantees that all weights of $\widehat{D}$ must have weight a multiple of 4. Thus $d_{min}(\widehat{D}) \geq 8$. We have equality, since each $\mathbf{c}_{0,j}$ has weight 8.

Puncturing $\widehat{D}$ at any position, we find a code of minimum distance at least 7; so every binary Golay code is a 3-error-correcting code. To complete (2) and the theorem, we check equality in the Sphere Packing Condition 2.2.5 for a binary Golay code:

$$\begin{aligned} 2^{23} &\geq 2^{12} \sum_{i=0}^{3} \binom{23}{i} (2-1)^i \\ &= 2^{12}\left(\binom{23}{0} + + \binom{23}{1} + \binom{23}{2} + \binom{23}{3}\right) \\ &= 2^{12}(1 + 23 + 253 + 1771) \\ &= 2^{12}(2048) = 2^{12}2^{11} = 2^{23}\,. \qquad \square \end{aligned}$$

7.3.3 Perfect codes

Although we will not at present devote much time to perfect codes, we emphasize the speciality of the Golay codes by reporting

(7.3.4) THEOREM. *(*TIETÄVÄINEN AND VAN LINT, 1971.*) A perfect e-error-correcting code C of length n over $\mathbb{F}_q$ satisfies one of:*

(1) $|C| = 1,\ e = n$;
(2) $|C| = q^n,\ e = 0$;
(3) $|C| = 2,\ q = 2,\ n = 2e + 1$;
(4) $|C| = 3^6,\ q = 3,\ e = 2,\ n = 11$;
(5) $|C| = 2^{12},\ q = 2,\ e = 3,\ n = 23$;
(6) $|C| = q^{n-r},\ e = 1,\ n = (q^r - 1)/(q - 1)$, *any* $r > 0$. $\square$

Notice that we make no assumption of linearity.

The codes of (1) and (2) are called trivial perfect codes. The repetition codes are examples for (3) and are nearly trivial. The Golay codes are examples in (4) and (5), and the Hamming codes occur under (6).

The codes of (1) through (5) are unique up to affine equivalence. This is easy to prove for (1) through (3) but difficult for the Golay codes. In most cases there exist perfect codes with the same parameters as a Hamming code but not affine equivalent to a Hamming code.

Best and Hong have proven that Theorem 7.3.4 remains valid for all finite alphabets A, not just those of prime power order, provided $e \geq 3$.

There are two basic tools in the proof of such theorems. One is the Sphere Packing Condition 2.2.5, in the form

$$\sum_{i=0}^{e} \binom{n}{i} (q-1)^i \mid q^n \,,$$

of particular value when the alphabet size q is a prime power and $e > 1$. Indeed the only solution to this equation for $q\,(\leq 100)$ a prime power with $n \leq 1000$ and $1 < e \leq 1000$, other than the ones implied by the existence of the perfect codes above, is

$$1 + \binom{90}{1} + \binom{90}{2} = 2^{12} \,,$$

which would correspond to a perfect binary 2-error-correcting code of length 90 (but see Problem 7.3.5 below).

The second main tool for proving nonexistence of perfect codes is Lloyd's Theorem, which is proven below as Theorem 9.4.9. This is a deep result saying that a certain polynomial, determined entirely by the parameters n, q, and e, must have all its roots positive integers in order for there to exist a corresponding perfect code. The analysis of the roots of the Lloyd polynomial is delicate but far reaching. As q has more prime factors, the Sphere Packing Condition becomes less restrictive; so Best and Hong's proof must rely almost entirely on Lloyd's theorem.

As an example of the kind of argument that goes into Theorem 7.3.4 and its relatives, we present the special case of binary, perfect 2-error-correcting codes as Theorem 9.4.11 below.

(7.3.5) PROBLEM. *(a) In a binary perfect e error-correcting code of length n we must have $n+1$ a multiple of $e+1$. (*HINT: *Assume that the code contains the* **0***-vector. Consider the $n-e$ words of weight $e+1$, having common ones in a fixed set of e coordinate positions, and the distribution of these words into spheres of radius e around codewords.)*

(b) Prove that a perfect binary 2-error-correcting code of length 90 does not exist.

(7.3.6) PROBLEM. *Prove that a binary, perfect 1-error-correcting code of length 7 is a coset of a Hamming code.*

(7.3.7) PROBLEM. *Let C be a binary, perfect 1-error-correcting code of length n that contains* **0***.*

*(a) Prove that C contains $n(n-1)/6$ codewords of weight 3. (*HINT: *Every word of weight 2 is inside a unique sphere of radius 1 around a codeword of weight 3.)*

*(b) Prove that C contains $n(n-1)(n-3)/24$ codewords of weight 4. (*HINT: *Every word of weight 3 is either a codeword or is inside a unique sphere of radius 1 around a codeword of weight 4.)*

(7.3.8) PROBLEM. *Explain how, in theory, one could find recursively the number of codewords of any fixed weight in the perfect e-error-correcting code C (containing* **0***) in terms of e, the length n, and the size q of the alphabet.*

7.4 Subfield subcodes

An expanded code is longer than its parent code but has the same number of codewords. Subfield subcodes have the same length but are smaller than their parents.

Again let the field F have a subfield K, and let C be a code of length n over F. Then the *subfield subcode* $C|_K$ equals $C \cap K^n$, the set of those codewords of C all of whose coordinate entries belong to the subfield K. As before, the concept makes sense for nonlinear codes, but we shall concentrate on the linear case.

subfield subcode

Of course it initially seems possible that a subfield subcode will be too small to be of use. For linear C, the subcode $C|_K$ contains the $\mathbf{0}$-vector, but does it contain anything else? We shall respond to this by proving that, for H a check matrix for C, the matrix $\check{H}$ is a control matrix for $C|_K$. This will give us an upper bound for the redundancy of $C|_K$ and so a lower bound for its dimension.

The next lemma is used to prove this observation. As before we let $e_1, \ldots, e_m$ a basis for F over K. The $a_{*,j} \in K$ are the entries of column $\check{\alpha}_j$ of the matrix $\check{\boldsymbol{\alpha}}$ and the vector $\boldsymbol{\alpha}^{[i]}$ is row i of the matrix.

(7.4.1) LEMMA. *For $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in F^n$, let $\check{\alpha}_j = (a_{1,j}, \ldots, a_{m,j})^\top$ (for $1 \le j \le n$) and $\boldsymbol{\alpha}^{[i]} = (a_{i,1}, a_{i,2}, \ldots, a_{i,n})$ (for $1 \le i \le m$). For $\mathbf{b} = (b_1, \ldots, b_n) \in K^n$,*

$$\boldsymbol{\alpha} \cdot \mathbf{b} = 0 \text{ in } F^n$$

if and only if

$$\boldsymbol{\alpha}^{[i]} \cdot \mathbf{b} = 0 \text{ in } K^n, \quad \text{for all } 1 \le i \le m\,.$$

PROOF.

$$\begin{array}{lll}
\boldsymbol{\alpha} \cdot \mathbf{b} = 0 & \text{iff} & \sum_{j=1}^n \alpha_j b_j = 0 \\
& \text{iff} & \sum_{j=1}^n (\sum_{i=1}^m a_{i,j} e_i) b_j = 0 \\
& \text{iff} & \sum_{i=1}^m (\sum_{j=1}^n a_{i,j} b_j) e_i = 0 \\
& \text{iff} & \sum_{j=1}^n a_{i,j} b_j = 0\,, \quad \text{for all } i \\
& \text{iff} & \boldsymbol{\alpha}^{[i]} \cdot \mathbf{b} = 0\,, \quad \text{for all } i. \qquad \square
\end{array}$$

Let H be a check (or control) matrix for the code C over F. Thus

$$\mathbf{x} \in C \text{ if and only if } H\mathbf{x}^\top = \mathbf{0}\,.$$

For a vector $\mathbf{b}$ with all its entries from K, we then have

$$\mathbf{b} \in C|_K \text{ if and only if } H\mathbf{b}^\top = \mathbf{0}\,,$$

which, by Lemma 7.4.1, is equivalent to

$$\text{for } \mathbf{b} \in K^n,\ \mathbf{b} \in C|_K \text{ if and only if } \check{H}\mathbf{b}^\top = \mathbf{0}\,.$$

Therefore $\check{H}$ is a control matrix for $C|_K$, as claimed.

(7.4.2) THEOREM. *If C is a linear $[n,k,d]$ code over F, then the subfield subcode $C|_K = C \cap K^n$ is a $[n, k' \geq n - mr, d' \geq d]$ code over K, where $r = n - k$.*

PROOF. If $\mathbf{a}, \mathbf{b} \in C|_K$ and $t, s \in K$, then $t\mathbf{a} + s\mathbf{b}$ is in K^n, as all entries are from K, and is in C, since C is linear over $F \geq K$. Therefore $t\mathbf{a} + s\mathbf{b} \in C|_K$, and the subfield subcode is linear.

Clearly $C|_K$ has length n. Since it is contained within the linear code C, we must have $d_{min}(C|_K) \geq d_{min}(C)$. It remains to verify the bound on its dimension. The redundancy of C is $n-k = r$, and that is the number of rows in a check matrix H for C. We have above constructed from a H a control matrix $\check{H}$ for $C|_K$, having m rows for each row of H. We can get a check matrix for $C|_K$ by discarding any unneeded rows from $\check{H}$. Thus the redundancy of $C|_K$ is at most mr, hence its dimension is at least $n - mr$, as claimed. □

We shall see in the next section that the bounds on dimension and distance in the theorem can be met and can be exceeded.

7.5 Alternant codes

alternant code
strict

If $GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is a generalized Reed-Solomon code over the field F and K is a subfield of F, then the subfield subcode $K^n \cap GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is an *alternant code*. The code is *strict* if no α_i equals 0. Clearly alternant codes can be decoded as GRS codes, but a new type of decoding default is possible — decoding to a codeword in the parent GRS code but not in the child alternant code. Recall that the strict generalized Reed-Solomon codes were somewhat easier to decode than those that are not strict.

An immediate consequence of Theorem 7.4.2 is

(7.5.1) THEOREM. *The alternant code $K^n \cap GRS_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is a $[n, k', d']$ linear code over K with $k' \geq n - (n-k)m$ and $d' \geq n - k + 1$.* □

In our earlier work on generalized Reed-Solomon codes, the scaling vector $\mathbf{v}$ played little role. It enlarged the family of codes to the point that we could prove, in Theorem 5.1.6, that the dual of a generalized Reed-Solomon codes is also a generalized Reed-Solomon code. Other than that, it has been benign; and in most of our decoding examples we assumed it to be $\mathbf{1}$, the vector of 1's. Now in the study of alternant codes, the scaling vector $\mathbf{v}$ comes to life. Different choices produce different codes.

Let α be a primitive element in the field $\mathbb{F}_{2^m}$, and set

$$\boldsymbol{\alpha} = (1, \alpha, \alpha^2, \ldots, \alpha^j, \ldots, \alpha^{n-1}),$$

where $n = 2^m - 1$ is the order of α and the length of the vector $\boldsymbol{\alpha}$. Let $\mathbf{1}$ be the vector of length n consisting of n entries 1. Then by Theorem 5.1.6 and Problem 5.1.5(c) (or direct calculation)

$$GRS_{n,a}(\boldsymbol{\alpha}, \boldsymbol{\alpha})^{\perp} = GRS_{n,b}(\boldsymbol{\alpha}, \mathbf{1})$$

whenever $a + b = n$.

We will consider some related subfield subcodes. In doing this, choose as $\mathbb{F}_2$-basis for $\mathbb{F}_{2^m}$ the decreasing powers of α:

$$e_1 = \alpha^{m-1}, \ldots, e_i = \alpha^{m-i}, \ldots, e_m = 1\,.$$

The code $GRS_{n,n-1}(\boldsymbol{\alpha}, \boldsymbol{\alpha})$ has dimension $n-1$ and minimum distance $2 = n-(n-1)+1$. It has as check matrix H the canonical generator matrix of its dual $GRS_{n,1}(\boldsymbol{\alpha}, \mathbf{1})$, a code of dimension 1 spanned by the vector $\mathbf{1}$. Therefore

$$H = \mathbf{1} \text{ and } \check{H} = (\check{1}, \check{1}, \ldots, \check{1})\,,$$

a matrix whose first $m-1$ rows are $\mathbf{0}$ and whose last row is $\mathbf{1}$. The subfield subcode $\mathbb{F}_2^n \cap GRS_{n,n-1}(\boldsymbol{\alpha}, \boldsymbol{\alpha})$ therefore has, as check matrix, the single vector $\mathbf{1}$ and is revealed as the parity check code of length n, also of minimum distance 2.

On the other hand, the code $GRS_{n,n-1}(\boldsymbol{\alpha}, \mathbf{1})$ also has dimension $n-1$ and minimum distance 2 but has as check matrix L the canonical generator matrix $\boldsymbol{\alpha}$ of its dual $GRS_{n,1}(\boldsymbol{\alpha}, \boldsymbol{\alpha})$. We have

$$L = \boldsymbol{\alpha} \text{ and } \check{L} = (\check{1}, \check{\alpha}, \ldots, \check{\alpha}^j, \ldots, \check{\alpha}^{n-1})\,.$$

Now the subfield subcode $\mathbb{F}_2^n \cap GRS_{n,n-1}(\boldsymbol{\alpha}, \mathbf{1})$ has, as check matrix, the matrix $\check{L}$ in which each nonzero m-tuple appears exactly once as a column $\check{\alpha}^j$, for the appropriate j. The subfield subcode $\mathbb{F}_2^n \cap GRS_{n,n-1}(\boldsymbol{\alpha}, \mathbf{1})$ is thus seen to be a binary Hamming code.

In summary, the alternant codes

$$\mathbb{F}_2^n \cap GRS_{n,n-1}(\boldsymbol{\alpha}, \boldsymbol{\alpha}) \text{ and } \mathbb{F}_2^n \cap GRS_{n,n-1}(\boldsymbol{\alpha}, \mathbf{1})\,,$$

which differ only in the choice of scaling vector $\mathbf{v}$, are very different codes. The first is a parity check code. Its dimension is $n-1\,(> n-m)$ and minimum distance is 2, meeting the lower bound of Theorem 7.5.1 (and Theorem 7.4.2). The second is a Hamming code. It has dimension $n-m$, meeting the bound of Theorem 7.5.1 (and Theorem 7.4.2), and minimum distance $3\,(> 2)$.

(7.5.2) PROBLEM. *We have shown above that certain binary Hamming codes arise as alternant codes. More generally, prove that all Hamming codes (binary or not) can be realized as alternant codes.*

(7.5.3) PROBLEM. *Prove that extended alternant codes (that is, the subfield subcodes coming from extended generalized Reed-Solomon codes) are strict alternant codes. In particular, all generalized Reed-Solomon codes can be realized as strict alternant codes.*

Chapter 8

Cyclic Codes

Among the first codes used practically were the cyclic codes which were generated using shift registers. It was quickly noticed by Prange that the class of cyclic codes has a rich algebraic structure, the first indication that algebra would be a valuable tool in code design.

The linear code C of length n is a *cyclic code* if it is invariant under a cyclic shift: cyclic code

$$\mathbf{c} = (c_0, c_1, c_2 \ldots, c_{n-2}, c_{n-1}) \in C$$

if and only if

$$\tilde{\mathbf{c}} = (c_{n-1}, c_0, c_1 \ldots, c_{n-3}, c_{n-2}) \in C\,.$$

As C is invariant under this single right cyclic shift, by iteration it is invariant under any number of right cyclic shifts. As a single left cyclic shift is the same as $n-1$ right cyclic shifts, C is also invariant under a single left cyclic shift and hence all left cyclic shifts. Therefore the linear code C is cyclic precisely when it is invariant under all cyclic shifts.

There are some obvious examples of cyclic codes. The $\mathbf{0}$-code is certainly cyclic as is F^n. Less trivially, repetition codes are cyclic. The binary parity check code is also cyclic, and this goes over to the sum-0 codes over any field.

Notice that this shift invariance criterion does not depend at all upon the code being linear. It is possible to define nonlinear cyclic codes, but that is rarely done. The history of cyclic codes as shift register codes and the mathematical structure theory of cyclic codes both suggest the study of cyclic invariance in the context of linear codes.

8.1 Basics

It is convenient to think of cyclic codes as consisting of polynomials as well as codewords. With every word

$$\mathbf{a} = (a_0, a_1, \ldots, a_i, \ldots, a_{n-2}, a_{n-1}) \in F^n$$

we associate the polynomial of degree less than n

$$a(x) = a_0 + a_1 x + \cdots + a_i x^i + \cdots + a_{n-1} x^{n-1} \in F[x]_n \,.$$

(We see here why in this chapter we index coordinates from 0 to $n-1$.) If $\mathbf{c}$ is a codeword of the code C, then we call $c(x)$ the associated *code polynomial*.

code polynomial

With this convention, the shifted codeword $\tilde{\mathbf{c}}$ has associated code polynomial

$$\tilde{c}(x) = c_{n-1} + c_0 x + c_1 x^2 + \cdots + c_i x^{i+1} + \cdots + c_{n-2} x^{n-1} \,.$$

Thus $\tilde{c}(x)$ is almost equal to the product polynomial $xc(x)$. More precisely,

$$\tilde{c}(x) = xc(x) - c_{n-1}(x^n - 1) \,.$$

Therefore $\tilde{c}(x)$ also has degree less than n and is equal to the remainder when $xc(x)$ is divided by $x^n - 1$. In particular

$$\tilde{c}(x) = xc(x) \pmod{x^n - 1} \,.$$

That is, $\tilde{c}(x)$ and $xc(x)$ are equal in the ring of polynomials $F[x] \pmod{x^n - 1}$, where arithmetic is done modulo the polynomial $x^n - 1$.

If $c(x)$ is the code polynomial associated with some codeword $\mathbf{c}$ of C, then we will allow ourselves to abuse notation by writing

$$c(x) \in C \,.$$

Indeed, if $f(x)$ is any polynomial of $F[x]$ whose remainder, upon division by $x^n - 1$, belongs to C then we may write

$$f(x) \in C \pmod{x^n - 1} \,.$$

With these notational conventions in mind, we see that our definition of the cyclic code C has the pleasing polynomial form

$$c(x) \in C \pmod{x^n - 1} \text{ if and only if } xc(x) \in C \pmod{x^n - 1} \,.$$

Since additional shifts do not take us out of the cyclic code C, we have

$$x^i c(x) \in C \pmod{x^n - 1} \,,$$

for all i. By linearity, for any $a_i \in F$,

$$a_i x^i c(x) \in C \pmod{x^n - 1}$$

and indeed

$$\sum_{i=0}^{d} a_i x^i \, c(x) \in C \pmod{x^n - 1} \,,$$

That is, for every polynomial $a(x) = \sum_{i=0}^{d} a_i x^i \in F[x]$, the product $a(x)c(x)$ (or more properly $a(x)c(x) \pmod{x^n - 1}$) still belongs to C. As linear C is certainly closed under polynomial addition, this says that the polynomial rendering of a cyclic code is precisely an ideal of the ring $F[x] \pmod{x^n - 1}$. This correspondence, first noted by Prange, opened the way for the application of algebra to cyclic codes.

(8.1.1) THEOREM. *Let $C \neq \{\mathbf{0}\}$ be a cyclic code of length n over F.*

(1) *Let $g(x)$ be a monic code polynomial of minimal degree in C. Then $g(x)$ is uniquely determined in C, and*

$$C = \{\, q(x)g(x) \,|\, q(x) \in F[x]_{n-r} \,\},$$

where $r = \deg(g(x))$. In particular, C has dimension $n - r$.

(2) *The polynomial $g(x)$ divides $x^n - 1$ in $F[x]$.*

PROOF. As $C \neq \{\mathbf{0}\}$, it contains non-zero code polynomials, each of which has a unique monic scalar multiple. Thus there is a monic polynomial $g(x)$ in C of minimal degree. Let this degree be r, unique even if $g(x)$ is not.

By the remarks preceding the theorem, the set of polynomials

$$C_0 = \{\, q(x)g(x) \,|\, q(x) \in F[x]_{n-r} \,\}$$

is certainly contained in C, since it is composed of those multiples of the code polynomial $g(x)$ with the additional property of having degree less than n. Under addition and scalar multiplication C_0 is an F-vector space of dimension $n - r$. The polynomial $g(x)$ is the unique monic polynomial of degree r in C_0.

To prove (1), we must show that every code polynomial $c(x)$ is an $F[x]$-multiple of $g(x)$ and so is in the set C_0. By the Division Algorithm A.2.5 we have

$$c(x) = q(x)g(x) + r(x)\,,$$

for some $q(x), r(x) \in F[x]$ with $\deg(r(x)) < r = \deg(g(x))$. Therefore

$$r(x) = c(x) - q(x)g(x)\,.$$

By definition $c(x) \in C$ and $q(x)g(x)$ is in C_0 (as $c(x)$ has degree less than n). Thus by linearity, the righthand side of this equation is in C, hence the remainder term $r(x)$ is in C. If $r(x)$ was non-zero, then it would have a monic scalar multiple belonging to C and of smaller degree. But this would contradict the original choice of $g(x)$. Therefore $r(x) = 0$ and $c(x) = q(x)g(x)$, as desired.

Next let

$$x^n - 1 = h(x)g(x) + s(x)\,,$$

for some $s(x)$ of degree less than $\deg(g(x))$. Then, as before,

$$s(x) = (-h(x))g(x) \pmod{x^n - 1}$$

belongs to C. Again, if $s(x)$ is not zero, then it has a monic scalar multiple belonging to C and of smaller degree than that of $g(x)$, a contradiction. Thus $s(x) = 0$ and $g(x)h(x) = x^n - 1$, as in (2). □

The polynomial $g(x)$ is called the *generator polynomial* for the code C. generator polynomial

The polynomial $h(x) \in F[x]$ determined by

$$g(x)h(x) = x^n - 1$$

check polynomial

is the *check polynomial* of C.

Under some circumstances it is convenient to consider $x^n - 1$ to be the generator polynomial of the cyclic code $\mathbf{0}$ of length n. Then by the theorem, there is a one-to-one correspondence between cyclic codes of length n and monic divisors of $x^n - 1$ in $F[x]$.

EXAMPLE. Consider length 7 binary cyclic codes. We have the factorization into irreducible polynomials

$$x^7 - 1 = (x-1)(x^3 + x + 1)(x^3 + x^2 + 1)\,.$$

Since we are looking at binary codes, all the minus signs can be replaced by plus signs:

$$x^7 + 1 = (x+1)(x^3 + x + 1)(x^3 + x^2 + 1)\,.$$

As there are 3 irreducible factors, there are $2^3 = 8$ cyclic codes (including $\mathbf{0}$ and $\mathbb{F}_2^7$). The 8 generator polynomials are:

$$\begin{array}{rl}
(i) & 1 = 1 \\
(ii) & x+1 = x+1 \\
(iii) & x^3 + x + 1 = x^3 + x + 1 \\
(iv) & x^3 + x^2 + 1 = x^3 + x^2 + 1 \\
(v) & (x+1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1 \\
(vi) & (x+1)(x^3 + x^2 + 1) = x^4 + x^2 + x + 1 \\
(vii) & (x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\
(viii) & (x+1)(x^3 + x + 1)(x^3 + x^2 + 1) = x^7 + 1
\end{array}$$

Here in (i) the polynomial 1 generates all $\mathbb{F}_2^7$. In (ii) we find the parity check code and in (vii) the repetition code . As mentioned before, in $(viii)$ we view the $\mathbf{0}$-code as being generated by $x^7 + 1$.

The polynomials of (iii) and (iv) have degree 3 and so generate $[7, 4]$ codes, which we shall later see are Hamming codes. The $[7, 3]$ codes of (v) and (vi) are the duals of the Hamming codes.

(8.1.2) PROBLEM. *How many cyclic codes of length* 8 *over* $\mathbb{F}_3$ *are there? Give a generator polynomial for each such code.*

(8.1.3) PROBLEM. *Prove that there is no cyclic code that is (equivalent to) an* $[8, 4]$ *extended binary Hamming code.*

(8.1.4) PROBLEM. *Let cyclic code* C *have generator polynomial* $g(x)$*. Prove that* C *is contained in the sum-*0 *code if and only if* $g(1) = 0$*.*

(8.1.5) PROBLEM. *Let* C *be a cyclic code. Let* C_- *be the code resulting from shortening* C *at a single position, and let* C^- *be the code resulting from puncturing* C *at a single position.*

(a) Give all C *for which* C_- *is cyclic.*

(b) Give all C *for which* C^- *is cyclic.*

The check polynomial earns its name by the following

(8.1.6) PROPOSITION. *If C is the cyclic code of length n with check polynomial $h(x)$, then*

$$C = \{\, c(x) \in F[x]_n \mid c(x)h(x) = 0 \pmod{x^n - 1} \,\}\,.$$

PROOF. The containment in one direction is easy. Indeed if $c(x) \in C$, then by Theorem 8.1.1 there is a $q(x)$ with $c(x) = q(x)g(x)$. But then

$$c(x)h(x) = q(x)g(x)h(x) = q(x)(x^n - 1) = 0 \pmod{x^n - 1}\,.$$

Now consider an arbitrary polynomial $c(x) \in F[x]_n$ with

$$c(x)h(x) = p(x)(x^n - 1), \text{ say }.$$

Then

$$\begin{aligned} c(x)h(x) &= p(x)(x^n - 1) \\ &= p(x)g(x)h(x)\,, \end{aligned}$$

hence

$$(c(x) - p(x)g(x))h(x) = 0\,.$$

As $g(x)h(x) = x^n - 1$, we do not have $h(x) = 0$. Therefore

$$\begin{aligned} c(x) - p(x)g(x) &= 0 \\ \text{and } \ c(x) &= p(x)g(x)\,, \end{aligned}$$

as desired. □

If we are in possession of a generator polynomial $g(x) = \sum_{j=0}^{r} g_j x^j$ for the cyclic code C, then we can easily construct a generator matrix for C. Consider

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r & 0 & 0 & \ldots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r \end{bmatrix}$$

The matrix G has n columns and $k = n - r$ rows; so the first row, row $\mathbf{g}_0$, finishes with a string of 1's of length $k - 1$. Each successive row is the cyclic shift of the previous row: $\mathbf{g}_i = \tilde{\mathbf{g}}_{i-1}$, for $i = 1, \ldots, k-1$. As $g(x)h(x) = x^n - 1$, we have

$$g_0 h_0 = g(0)h(0) = 0^n - 1 \neq 0\,.$$

In particular $g_0 \neq 0$ (and $h_0 \neq 0$). Therefore G is in echelon form (although likely not reduced). In particular the $k = \dim(C)$ rows of G are linearly independent. Clearly the rows of G belong to C, so G is indeed a generator matrix for C, sometimes called the *cyclic generator matrix* of C.

cyclic generator matrix

For instance, if C is a $[7,4]$ binary cyclic code with generator polynomial $1+x+x^3$, then the cyclic generator matrix is

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

cyclic encoding

Given the cyclic generator matrix G, *cyclic encoding* is the process of encoding the message k-tuple $\mathbf{m} = (m_0, \ldots, m_{k-1})$ into the codeword $\mathbf{c} = \mathbf{m}G$.

message polynomial

At the polynomial level, this corresponds to encoding the *message polynomial* $m(x) = \sum_{i=0}^{k-1} m_i x^i$ into the code polynomial $c(x) = m(x)g(x)$.

Since the cyclic generator G is in echelon form, the first k coordinate positions form an information set. Therefore cyclic C has a standard generator matrix, although the cyclic generator matrix is almost never standard (or even systematic).

(8.1.7) PROBLEM. *(a) Describe all situations in which the cyclic generator matrix for a cyclic code is the standard generator matrix.*

(b) Describe all situations in which the cyclic generator matrix for a cyclic code is systematic.

We next present for cyclic C a linear encoding method corresponding to the standard generator matrix. Namely

$$\mathbf{m} = (m_0, \ldots, m_{k-1}) \mapsto \mathbf{c} = (m_0, \ldots, m_{k-1}, -s_0, -s_1, \ldots, -s_{r-1}),$$

where $s(x) = \sum_{j=0}^{r-1} s_j x^j$ is the remainder upon dividing $x^r m(x)$ by $g(x)$. That is,

$$x^r m(x) = q(x)g(x) + s(x),$$

with $\deg(s(x)) < \deg(g(x)) = r$. To see that this is the correct standard encoding, first note that

$$x^r m(x) - s(x) = q(x)g(x) = b(x) \in C$$

with corresponding codeword

$$\mathbf{b} = (-s_0, -s_1, \ldots, -s_{r-1}, m_0, \ldots, m_{k-1}).$$

As this is a codeword of cyclic C, every cyclic shift of it is also a codeword. In particular the $\mathbf{c}$ given above is found after k right shifts. Thus $\mathbf{c}$ is a codeword of C. Since C is systematic on the first k positions, this codeword is the only one with $\mathbf{m}$ on those positions and so is the result of standard encoding. To construct the standard generator matrix itself, we encode the k different k-tuple messages $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ of weight 1 corresponding to message polynomials x^i, for $0 \le i \le k-1$. These are the rows of the standard generator matrix.

When we try this for the $[7,4]$ binary cyclic code with generator x^3+x+1 (so $r = 7-4 = 3$), we find, for instance,

$$x^3 x^2 = (x^2+1)(x^3+x+1) + (x^2+x+1)$$

so that the third row of the standard generator matrix, corresponding to message polynomial x^2, is

$$(m_0, m_1, m_2, m_3, -s_0, -s_1, -s_2) = (0, 0, 1, 0, 1, 1, 1)\,.$$

Proceeding in this way, we find that the standard generator matrix is

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

By Problem 4.1.9, C is a Hamming code (although this can also be checked easily by hand).

This process of *systematic encoding* for cyclic codes is important in practice, since a machine can be transmitting the information symbols from $\mathbf{m}$ during the time it is calculating the check symbols s_j. systematic encoding

(8.1.8) PROBLEM. *(a) Find the cyclic and standard generator matrices for the* $[7, 4]$ *binary cyclic code* D *with generator polynomial* $x^3 + x^2 + 1$.

(b) Find the cyclic and standard generator matrices for the $[15, 11]$ *binary cyclic code* E *with generator polynomial* $x^4 + x + 1$.

(c) Prove that D *and* E *are Hamming codes.*

A code equivalent to a cyclic code need not be cyclic itself. For instance, there are 70 distinct binary $[7, 4]$ Hamming codes; but, as we saw in the example above, only two of them are cyclic.

One permutation does take cyclic codes to cyclic codes. The *reverse code* $C^{[-1]}$ of a cyclic code C, gotten by reversing each codeword, is still cyclic. We have reverse code

$$(c_0, c_1, \ldots, c_i, \ldots, , c_{n-1}) \in C \quad \text{iff} \quad (c_{n-1}, \ldots, c_{n-1-i}, \ldots, c_1, c_0) \in C^{[-1]}\,.$$

In polynomial notation, this becomes

$$c(x) \in C \quad \text{iff} \quad x^{n-1}c(x^{-1}) \in C^{[-1]}\,.$$

For the polynomial $p(x)$ of degree d, we let its *reciprocal polynomial* be given by reciprocal polynomial

$$p^{[-1]}(x) = \sum_{i=0}^{d} p_{d-i}x^i = x^d p(x^{-1})\,.$$

The roots of the reciprocal polynomial are the reciprocals of the nonzero roots of the original polynomial.

(8.1.9) LEMMA. *If* $g(x)$ *generates cyclic* C*, then* $g_0^{-1}g^{[-1]}(x)$ *generates* $C^{[-1]}$*, the reverse code of* C.

PROOF. Starting from the cyclic generator matrix for C, we reverse all the rows and then write them from bottom to top. The result is

$$\begin{bmatrix} g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 & 0 & 0 & \dots & 0 \\ 0 & g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 \end{bmatrix}.$$

The rows of this matrix certainly belong to $C^{[-1]}$. As before, they are linearly independent since $g_0 \neq 0$. Therefore we have a generator matrix for $C^{[-1]}$. Its first row visibly corresponds to a non-zero code polynomial of degree less than r, which is seen to be $g^{[-1]}(x)$. By Theorem 8.1.1 the monic scalar multiple $g_0^{-1}g^{[-1]}(x)$ is the generator polynomial. (In fact, we have a scalar multiple of the cyclic generator matrix for $C^{[-1]}$.) □

It is easy to see that the dual of a cyclic code C is again a cyclic code. Proposition 8.1.6 suggests that the dual is associated with the check polynomial of C.

Let the cyclic code C of length n have generator polynomial $g(x)$ of degree r and check polynomial $h(x)$ of degree $k = n - r = \dim C$. As $h(x)$ is a divisor of $x^n - 1$, it is the generator polynomial for a cyclic code D of length n and dimension $n - k = n - (n - r) = r$. We have

$$C = \{\, q(x)g(x) \,|\, q(x) \in F[x]_k \,\}$$

and

$$D = \{\, p(x)h(x) \,|\, p(x) \in F[x]_r \,\}.$$

Let $c(x) = q(x)g(x) \in C$, so that $\deg(q(x)) \leq k - 1$; and let $d(x) = p(x)h(x) \in D$, so that $\deg(p(x)) \leq r - 1$. Consider

$$\begin{aligned} c(x)d(x) &= q(x)g(x)p(x)h(x) \\ &= q(x)p(x)(x^n - 1) \\ &= s(x)(x^n - 1) \\ &= s(x)x^n - s(x), \end{aligned}$$

where $s(x) = q(x)p(x)$ with

$$\deg(s(x)) \leq (k - 1) + (r - 1) = r + k - 2 = n - 2 < n - 1.$$

Therefore the coefficient of x^{n-1} in $c(x)d(x)$ is 0. If $c(x) = \sum_{i=0}^{n-1} c_i x^i$ and $d(x) = \sum_{j=0}^{n-1} d_j x^j$, then in general the coefficient of x^m in $c(x)d(x)$ is $\sum_{i+j=m} c_i d_j$. In particular, the two determinations of the coefficient of x^{n-1} in $c(x)d(x)$ give

$$0 = \sum_{i+j=n-1} c_i d_j$$

$$\begin{aligned} &= \sum_{i=0}^{n-1} c_i d_{n-i} \\ &= c_0 d_{n-1} + c_1 d_{n-2} + \cdots + c_i d_{n-i} + \cdots + c_{n-1} d_0 \\ &= \mathbf{c} \cdot \mathbf{d}^* \,. \end{aligned}$$

where

$$\mathbf{c} = (c_0, c_1, \ldots, c_i, \ldots, c_{n-1}) \text{ and } \mathbf{d}^* = (d_{n-1}, d_{n-2}, \ldots, d_{n-i}, \ldots, d_0) \,.$$

That is, each codeword $\mathbf{c}$ of C has dot product 0 with the reverse of each codeword $\mathbf{d}$ of D.

Therefore $C^\perp$ contains $D^{[-1]}$. Also

$$\dim(C^\perp) = n - \dim(C) = n - k = r = n - \deg(h^{[-1]}(x)) = \dim(D^{[-1]}) \,,$$

so from Lemma 8.1.9 we conclude

(8.1.10) THEOREM. *If C is the cyclic code of length n with check polynomial $h(x)$, then $C^\perp$ is cyclic with generator polynomial $h_0^{-1} h^{[-1]}(x)$.* □

8.2 Cyclic *GRS* codes and Reed-Solomon codes

For α a primitive n^{th} root of unity in the field F, set

$$\begin{aligned} \boldsymbol{\alpha}^{(a)} &= ((\alpha^0)^a, \ldots, (\alpha^j)^a, \ldots, (\alpha^{n-1})^a) \\ &= ((\alpha^a)^0, \ldots, (\alpha^a)^j, \ldots, (\alpha^a)^{n-1}) \,. \end{aligned}$$

In particular, $\boldsymbol{\alpha} = \boldsymbol{\alpha}^{(1)}$ and $\boldsymbol{\alpha}^{(0)} = \mathbf{1}$, the all 1-vector.

The basic observation is that

$$\begin{aligned} \tilde{\boldsymbol{\alpha}}^{(a)} &= ((\alpha^{n-1})^a, (\alpha^0)^a, \ldots, (\alpha^j)^a, \ldots, (\alpha^{n-2})^a) \\ &= \alpha^{-a}((\alpha^0)^a, (\alpha^1)^a, \ldots, (\alpha^j)^a, \ldots, (\alpha^{n-1})^a) \\ &= \alpha^{-a} \boldsymbol{\alpha}^{(a)} \,. \end{aligned}$$

Thus a cyclic shift of $\boldsymbol{\alpha}^{(a)}$ is always a scalar multiple of $\boldsymbol{\alpha}^{(a)}$.

(8.2.1) PROPOSITION. *$GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$ is cyclic.*

PROOF. For $0 \le i \le k-1$ and $0 \le j \le n-1$, the (i,j)-entry of the canonical generator matix is

$$\begin{aligned} v_j \alpha_j^i &= (\alpha^j)^a (\alpha^j)^i \\ &= \alpha^{ja} \alpha^{ji} = (\alpha^j)^{a+i} \,. \end{aligned}$$

Therefore the canonical generator matrix has as rows the k codewords $\boldsymbol{\alpha}^{(a+i)}$, for $i = 0, \ldots, k-1$. We have seen above that shifting any of these only gives scalar multiples, so the code itself is invariant under shifting. □

Reed-Solomon code
primitive
narrow-sense

A cyclic code $GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$ as in Proposition 8.2.1 is a *Reed-Solomon code*. It is said to be *primitive* if $n = |F| - 1$ and of *narrow-sense* if $a = 0$ (so that $\mathbf{v} = \boldsymbol{\alpha}^{(a)} = \mathbf{1}$).

(8.2.2) LEMMA. *If $\alpha^n = 1$ and $\boldsymbol{\alpha} = (\alpha^0, \ldots, \alpha^{n-1})$, then*

$$GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})^\perp = GRS_{n,n-k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(1-a)}) .$$

PROOF. By Theorem 5.1.6

$$GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})^\perp = GRS_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u}) ,$$

where, for $0 \le j \le n-1$ and $\mathbf{v} = \boldsymbol{\alpha}^{(a)}$, we have $u_j = v_j^{-1} L_j(\alpha^j)^{-1}$. By Problem 5.1.5(c), $L_j(\alpha^j) = n(\alpha^j)^{-1} (\neq 0)$. Thus

$$\begin{aligned} u_j &= ((\alpha^j)^a)^{-1}(n(\alpha^j)^{-1})^{-1} \\ &= n^{-1}\alpha^{-ja}\alpha^j \\ &= n^{-1}(\alpha^j)^{1-a} \end{aligned}$$

Therefore $\mathbf{u} = n^{-1}\boldsymbol{\alpha}^{(1-a)}$, so by Problem 5.1.3(a)

$$\begin{aligned} GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})^\perp &= GRS_{n,n-k}(\boldsymbol{\alpha}, n^{-1}\boldsymbol{\alpha}^{(1-a)}) \\ &= GRS_{n,n-k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(1-a)}) \end{aligned}$$

as desired. □

(8.2.3) THEOREM. *An $[n, k]$ Reed-Solomon code over F is a cyclic code with generator polynomial*

$$\prod_{j=1}^{t} (x - \alpha^{j+b})$$

where $t = n - k$, the integer b is a fixed constant, and α is a primitive n^{th} root of unity in F. This Reed-Solomon code is primitive if $n = |F| - 1$ and narrow-sense if $b = 0$.

PROOF. Let $C = GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$. The rows of the canonical generator matrix of the dual code $C^\perp$ are, by Lemma 8.2.2 and a previous calculation, the vectors $\boldsymbol{\alpha}^{(j-a)}$, for $1 \le j \le t$. Therefore, for $\mathbf{c} = (c_0, \ldots, c_i, \ldots, c_{n-1})$ and $c(x) = \sum_{i=0}^{n-1} c_i x^i$,

$$\begin{aligned} \mathbf{c} \in C \quad &\text{iff} \quad \mathbf{c} \cdot \boldsymbol{\alpha}^{(j-a)} = 0, \quad 1 \le j \le t \\ &\text{iff} \quad \sum_{i=0}^{n-1} c_i (\alpha^i)^{j-a} = 0, \quad 1 \le j \le t \\ &\text{iff} \quad \sum_{i=0}^{n-1} c_i (\alpha^{j-a})^i = 0, \quad 1 \le j \le t \\ &\text{iff} \quad c(\alpha^{j-a}) = 0, \quad 1 \le j \le t . \end{aligned}$$

Thus, writing cyclic C in terms of polynomials, we have by Lemma A.2.8

$$\begin{aligned} c(x) \in C \quad &\text{iff} \quad c(\alpha^{j-a}) = 0, \quad 1 \le j \le t \\ &\text{iff} \quad \prod_{j=1}^{t} (x - \alpha^{j+b}) \text{ divides } c(x)\,, \end{aligned}$$

for $b = -a$. As $\prod_{j=1}^{t}(x - \alpha^{j+b})$ is monic and has degree $t = n - k$, it is the generator polynomial of C by Theorem 8.1.1.

Also α is a primitive element of F when $n = |F| - 1$; and C is narrow-sense when $a = 0$, that is, when $b = -a = 0$. □

In most places, the statement of Theorem 8.2.3 is taken as the definition of a Reed-Solomon code. It is then proven that such a code is MDS with $d_{min} = t + 1 = n - k + 1$. Our development is somewhat closer to the original presentation of Reed and Solomon from 1960.

(8.2.4) PROBLEM. *Prove that $EGRS_{q+1,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$, where $|F| = q$, is monomially equivalent to a cyclic code when q is even and to a negacyclic code when q is odd. Here a code C is* negacyclic *provided* negacyclic

$$(c_0, c_1, c_2 \ldots, c_{n-2}, c_{n-1}) \in C$$

if and only if

$$(-c_{n-1}, c_0, c_1 \ldots, c_{n-3}, c_{n-2}) \in C\,.$$

*(*HINT: *See Theorem 6.3.4.)*

8.3 Cylic alternant codes and BCH codes

Let $K \le F$ be fields. Starting with the Reed-Solomon code $GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$ over F, the cyclic, alternant code $C = K^n \cap GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$ is called a *BCH code of designed distance $t + 1$*, where $t = n - k$. C is *primitive* if $n = |F| - 1$ and *narrow-sense* if $a = 0$ (that is to say, $\mathbf{v} = \mathbf{1}$).

(8.3.1) THEOREM. *A BCH code C of length n and designed distance $t + 1$ over K is a cyclic code composed of all those code polynomials $c(x) \in K[x]$ of degree less than n satisfying*

$$c(\alpha^{b+1}) = c(\alpha^{b+2}) = c(\alpha^{b+3}) = \cdots = c(\alpha^{b+t}) = 0\,,$$

where b is a fixed integer and α is a primitive n^{th} root of unity in the field $F \ge K$. The code is primitive if $n = |F| - 1$ and is narrow-sense if $b = 0$.

The code C is linear and cyclic with generator polynomial

$$\text{lcm}_{1 \le j \le t} \{m_{\alpha^{j+b}, K}(x)\}\,.$$

It has minimum distance at least $t + 1$ and dimension at least $n - mt$, where $m = \dim_K F$.

PROOF. The first paragraph is an immediate consequence of Theorem 8.2.3 and the definitions. As C is the alternant code $K^n \cap GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^{(a)})$ it is, by Theorem 7.5.1, linear of minimum distance at least $n-k+1 = t+1$ and dimension at least $n - m(n-k) = n - mt$. The form taken by the generator polynomial follows from the first paragraph and Lemma A.3.19 of the Appendix. □

As with Reed-Solomon codes, the first paragraph of this theorem consists of the usual definition of a BCH code. Indeed, that is essentially the original definition as given by Bose and Ray-Chadhuri (1960) and Hocquenghem (1959). (The codes were then given the somewhat inaccurate acronym as name.) It then must be proven that the designed distance of a BCH code gives a lower bound for the actual minimum distance. In many places Reed-Solomon codes are defined as those BCH codes in which the fields F and K are the same. Historically, the two classes of codes were discovered independently and the connections only noticed later.

Sometimes one takes a different view of Theorem 8.3.1, viewing it instead as a general bound on cyclic codes in terms of root patterns for the generator polynomial.

(8.3.2) COROLLARY. *(BCH BOUND.) Let C be a cyclic code of length n over K with generator polynomial $g(x)$. Suppose that $g(\alpha^{j+b}) = 0$, for some fixed b and $1 \leq j \leq t$, where α is a primitive n^{th} root of unity in the field $F \geq K$. Then $d_{min}(C) \geq t+1$.*

PROOF. In this case, C is a subcode of a BCH code with designed distance $t+1$. □

This corollary admits many generalizations, the general form of which states that a certain pattern of roots for the generator polynomial of a cyclic code implies a lower bound for the minimum distance.

(8.3.3) PROBLEM. *Assume that the cyclic code C has generator polynomial $g(x)$ with $g(1) \neq 0$. Prove that $(x-1)g(x)$ is the generator polynomial of the sum-0 subcode of C (those codewords of C whose coordinate entries sum to 0).*

The last sentence in the theorem gives us two lower bounds for BCH codes, one for the minimum distance (the BCH bound) and one for the dimension. As we prefer large distance and dimension, we would hope to find situations in which one or both of these bounds are not met exactly. For any cyclic code, the generator polynomial has degree equal to the redundancy of the code. In Theorem 8.3.1 that degree/redundancy is bounded above by mt. This bound will be met exactly if and only if each of the minimal polynomials $m_{\alpha^{j+b},K}(x)$ has the maximum possible degree m and, additionally, all of these polynomials, for $1 \leq j \leq t$, are distinct. This sounds an unlikely event but can, in fact, happen. Conversely we often can make our choices so as to guarantee that the degree of the generator is dramatically less than this maximum. We shall see

below that the two bounds of the theorem are independent and can be either met or beaten, depending upon the specific circumstances. (Both bounds are tight for Reed-Solomon codes, but there are other cases as well where this happens.)

(8.3.4) COROLLARY. (1) *A binary, narrow-sense, primitive BCH code of designed distance* 2 *is a cyclic Hamming code.*

(2) *A binary, narrow-sense, primitive BCH code of designed distance* 3 *is a cyclic Hamming code.*

PROOF. Let $n = 2^m - 1$ and $K = \mathbb{F}_2 \leq \mathbb{F}_{2^m} = F$. Let α be a primitive element in $\mathbb{F}_{2^m}$ (so it has order n). Then the associated designed distance 2 code C_2 has generator polynomial

$$m(x) = m_\alpha(x) = m_{\alpha,\mathbb{F}_2}(x)$$

of degree m, the minimal polynomial of α over the prime subfield $\mathbb{F}_2$. The corresponding designed distance 3 code C_3 has generator polynomial

$$\mathrm{lcm}\{m_\alpha(x), m_{\alpha^2}(x)\}\,.$$

From Theorem A.3.20 we learn that $m_{\alpha^2}(x) = m_\alpha(x)$. Therefore this lcm is again equal to $m(x)$, and C_2 and C_3 both have generator polynomial $m(x)$ of degree m. Thus $C_2 = C_3$ has dimension $n - m = 2^m - 1 - m$ and minimum distance at least 3. It is therefore a Hamming code by Problem 4.1.3 or Problem 4.1.9. (Alternatively C_2 is, by Lemma 8.2.2, equal to the alternant code $\mathbb{F}_2^n \cap GRS_{n,1}(\boldsymbol{\alpha}, \boldsymbol{\alpha})^\perp$, which we have already identified as a Hamming code in Section 7.5.) □

From this corollary we learn that it is possible to find BCH codes with inequality in the distance bound (BCH bound) and equality in the dimension bound of Theorem 8.3.1 ($d_{min}(C_2) = 3 > 1 + 1$ and $\dim(C_2) = n - m \cdot 1$) and also BCH codes with equality in the distance bound and inequality in the dimension bound ($d_{min}(C_3) = 3 = 2 + 1$ and $\dim(C_3) = n - m > n - m \cdot 2$).

As in the corollary, narrow-sense codes frequently have better parameters than those that are not. For instance, in the situation of the corollary, the designed distance 2 code with $b = -1$ has generator polynomial $m_{\alpha^{1-1},\mathbb{F}_2}(x) = x - 1$. This code is the parity check code with d_{min} indeed equal to 2 and dimension $n - 1 (> n - m)$. When $n = 15$ (so that $m = 4$), the designed distance 2 code with $b = 2$ has generator polynomial

$$m_{\alpha^{1+2},\mathbb{F}_2}(x) = x^4 + x^3 + x^2 + x + 1 = (x^5 - 1)/(x - 1)\,,$$

since $(\alpha^3)^5 = \alpha^{15} = 1$. Therefore this code meets both bounds exactly, having dimension $11 = 15 - 4$ and minimum distance 2, as it contains the code polynomial $x^5 - 1$.

Consider next the binary, narrow-sense, primitive BCH code with length 15 and designed distance 5, defined using as primitive element α a root of the primitive polynomial $x^4 + x + 1$. The generator polynomial is, by Theorem 8.3.1,

$$g(x) = \mathrm{lcm}_{1 \leq j \leq 4}\{m_{\alpha^j}(x)\}\,.$$

By definition $m_\alpha(x) = x^4 + x + 1$, and we found $m_{\alpha^3}(x) = x^4 + x^3 + x^2 + x + 1$ above. By Theorem A.3.20 of the Appendix,

$$m_\alpha(x) = m_{\alpha^2}(x) = m_{\alpha^4}(x)\,,$$

therefore

$$\begin{aligned} g(x) &= m_\alpha(x) m_{\alpha^3}(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1\,. \end{aligned}$$

In particular, the code has dimension $15 - 8 = 7$, whereas the bound of Theorem 8.3.1 is useless, claiming only that the dimension is at least $15 - 4 \cdot 4 = -1$. Furthermore $g(x)$ itself has weight 5, so in this case the designed distance 5 code has minimum distance exactly 5. (Although the generator polynomial always has relatively low weight, in general it will not have the minimum weight. Still it is often worth checking.) We see again here the advantage of looking at narrow-sense codes. By Theorem A.3.20, whenever α^i is a root of $m(x)$, then α^{2i} is as well (in the binary case). In particular, the binary, narrow-sense, designed distance $2d$ code, given by roots α^j, for $1 \le j \le 2d - 1$, is also equal to the designed distance $2d + 1$ code, given by roots α^j, for $1 \le j \le 2d$, since α^d a root implies α^{2d} is as well. (We saw a particular case of this in Corollary 8.3.4.) Similar but weaker statements can be made for non-binary BCH codes by appealing to Theorem A.3.20 or the more general Problem A.3.21.

We also see that Theorem A.3.20 and Problem A.3.21 can be used effectively to calculate the parameters and generator polynomials of BCH codes. Consider next a binary, narrow-sense, primitive BCH code C of length 31 with designed distance 8. The previous paragraph already tells us that C is also the corresponding designed distance 9 code, but more is true. We have generator polynomial

$$\begin{aligned} g(x) &= \mathrm{lcm}_{1 \le j \le 8} \{m_{\alpha^j}(x)\} \\ &= m_\alpha(x) m_{\alpha^3}(x) m_{\alpha^5}(x) m_{\alpha^7}(x)\,, \end{aligned}$$

where α is an arbitrary but fixed primitive 31^{st} root of unity in $\mathbb{F}_{32}$. By Theorem A.3.20

$$\begin{aligned} m_\alpha(x) &= (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8)(x - \alpha^{15})\,; \\ m_{\alpha^3}(x) &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^{24})(x - \alpha^{17})\,; \\ m_{\alpha^5}(x) &= (x - \alpha^5)(x - \alpha^{10})(x - \alpha^{20})(x - \alpha^9)(x - \alpha^{18})\,; \\ m_{\alpha^7}(x) &= (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{28})(x - \alpha^{25})(x - \alpha^{19})\,. \end{aligned}$$

Therefore C has dimension $31 - 4 \cdot 5 = 11$. We also discover that we have gotten the roots α^9 and α^{10} 'for free', so that the designed distance 8(9) BCH code is actually equal to the designed distance 11 code (so in this case, neither of the bounds of Theorem 8.3.1 hold with equality). It is worth noting that we can calculate this dimension and improved BCH bound without explicitly finding

the generator polynomial. The calculations are valid no matter which primitive element α we choose. Examples below find explicit generator polynomials, using similar calculations based upon Theorem A.3.20.

The good fortune seen in the previous paragraph can often be dramatic. Berlekamp has noted that the binary, narrow-sense, primitive BCH code of length $2^{12}-1$ and designed distance 768 is equal to the corresponding designed distance 819 code. On the other hand, there are many situations where the BCH bound still does not give the true minimum distance. Roos, Van Lint, and Wilson have noted that the binary length 21 code with generator polynomial

$$m_\beta(x)m_{\beta^3}(x)m_{\beta^7}(x)m_{\beta^9}(x)\,,$$

which has as roots β^j (β a 21^{st} root of unity) for

$$j \in \{1,2,3,4,6,7,8,9,11,12,14,15,16,18\}\,,$$

has true minimum distance 8, whereas the BCH bound only guarantees distance at least 5.

EXAMPLES. (*i*) Let α be a root of the primitive polynomial $x^4+x^3+1 \in \mathbb{F}_2[x]$. What is the generator of the binary, narrow-sense, primitive BCH code C_1 of length 15 and designed distance 5?

The code is composed of all polynomials $c(x) \in \mathbb{F}_2[x]$ that have each of $\alpha^1, \alpha^2, \alpha^3, \alpha^4$ as a root. Therefore $c(x)$ is divisible by

$$m_\alpha(x) = (x-\alpha^1)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8) = x^4+x^3+1$$

and also by

$$m_{\alpha^3}(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^9) = x^4+x^3+x^2+x+1.$$

So C_1 has generator

$$g_1(x) = (x^4+x^3+1)(x^4+x^3+x^2+x+1) = x^8+x^4+x^2+x+1.$$

As $g_1(x)$ has degree 8, the code has dimension $15-8=7$. (Here again the generator has weight 5, so the minimal distance of this code equals 5.)

(*ii*) Let α be a root of the primitive polynomial $x^5+x^2+1 \in \mathbb{F}_2[x]$. What is the generator of the binary, narrow-sense, primitive BCH code C_2 of length 31 and designed distance 5?

Again the code is composed of all polynomials $c(x) \in \mathbb{F}_2[x]$ that have each of $\alpha^1, \alpha^2, \alpha^3, \alpha^4$ as a root. Therefore $c(x)$ is divisible by

$$m_\alpha(x) = (x-\alpha^1)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)(x-\alpha^{16}) = x^5+x^2+1$$

and also

$$m_{\alpha^3}(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^{24})(x-\alpha^{17}) = x^5+x^4+x^3+x^2+1.$$

So C_2 has generator

$$g_2(x) = (x^5+x^2+1)(x^5+x^4+x^3+x^2+1) = x^{10}+x^9+x^8+x^6+x^5+x^3+1.$$

As $g_2(x)$ has degree 10, the code has dimension $31 - 10 = 21$. (But notice that here the weight of the generator is larger than 5.)

(*iii*) Maintaining the notation of Example (*ii*), find the generator of the *BCH* code C_3 of length 31 with designed distance 7.

The code polynomials $c(x)$ must satisfy

$$c(\alpha^1) = c(\alpha^2) = c(\alpha^3) = c(\alpha^4) = c(\alpha^5) = c(\alpha^6) = 0.$$

In particular $c(x)$ must be a multiple of $g_2(x)$, calculated in the previous example. But $c(x)$ must also be a multiple of

$$m_{\alpha^5}(x) = (x-\alpha^5)(x-\alpha^{10})(x-\alpha^{20})(x-\alpha^9)(x-\alpha^{18}) = x^5+x^4+x^2+x+1.$$

(This calculation is done in detail in section A.3.3 of the Appendix on algebra.) Thus the generator $g_3(x)$ for C_3 is

$$g_2(x)(x^5+x^4+x^2+x+1) = x^{15}+x^{11}+x^{10}+x^9+x^8+x^7+x^5+x^3+x^2+x+1.$$

This code has dimension $31 - 15 = 16$.

(*iv*) Let β be a root of the irreducible but imprimitive polynomial $x^3 + 2x + 2 \in \mathbb{F}_3[x]$. Using β, find the generator polynomial of the ternary, narrow-sense *BCH* code D_1 of length 13 with designed distance 4.

The code polynomials must have as roots β and β^2. Thus they must be multiples of

$$m_\beta(x) = (x - \beta)(x - \beta^3)(x - \beta^9) = x^3 + 2x + 2$$

and of

$$m_{\beta^2}(x) = (x - \beta^2)(x - \beta^6)(x - \beta^5) = x^3 + x^2 + x + 2.$$

Therefore D_1 has generator

$$g_4(x) = (x^3 + 2x + 2)(x^3 + x^2 + x + 2) = x^6 + x^5 + x^2 + 1.$$

In particular the code has dimension $13 - 6 = 7$.

(8.3.5) PROBLEM. *Give the generator polynomial of the ternary, narrow-sense BCH code D_2 of length 13 with designed distance 5, using β of Example (iv) above as a primitive 13^{th} root of unity. What is the dimension of D_2?*

(8.3.6) PROBLEM. *Give the generator polynomial of the ternary, narrow-sense, primitive, BCH code D_3 of length 26 with designed distance 4, using as primitive element γ a root of the polynomial $x^3 + 2x + 1 \in \mathbb{F}_3[x]$. What is the dimension of D_3?*

(8.3.7) PROBLEM. *(a) What is the dimension of a binary, narrow-sense, primitive BCH code of length 63 and designed distance 17.*

(b) Does this code contain any codewords of weight 17? Explain your answer.

(8.3.8) PROBLEM. *Prove that a narrow-sense, primitive BCH code of length 24 over $\mathbb{F}_5$ with designed distance 3 has minimum distance 3 and dimension $20 = 24-2(3-1)$.*

8.4 Cyclic Hamming codes and their relatives

Cyclic binary Hamming codes and codes related to them are of particular interest.

(8.4.1) THEOREM. *For every m, there is a cyclic, binary Hamming code of redundancy m. Indeed any primitive polynomial of degree m in $\mathbb{F}_2[x]$ generates a cyclic Hamming code of redundancy m.*

PROOF. This is essentially equivalent to Corollary 8.3.4 (in view of Theorems A.3.8 and A.3.10 on the general structure and existence of finite fields). □

(8.4.2) THEOREM. *The polynomial $g(x) \in \mathbb{F}_2[x]$ generates a cyclic, binary Hamming code if and only if it is primitive.*

PROOF. In Theorem 8.4.1 we have seen that a binary primitive polynomial generates a cyclic Hamming code.

Now let C be a binary, cyclic Hamming code of length $2^m - 1 = n$. Let $g(x) = \prod_{i=1}^{r} g_i(x)$, where the $g_i(x)$ are distinct irreducible polynomials of degree m_i, so that $\sum_{i=1}^{r} m_i = m = \deg g(x)$. Then $g_i(x)$ divides $x^{n_i} - 1$ with $n_i = 2^{m_i} - 1$, hence $g(x)$ divides $x^{n_0} - 1$ where $n_0 = \prod_{i=1}^{r} n_i$. Now

$$n + 1 = 2^m - 1 + 1 = \prod_{i=1}^{r} 2^{m_i} = \prod_{i=1}^{r} (n_i + 1)\,.$$

If $r \neq 1$, then $n > n_0$ and $x^{n_0} - 1$ is a codeword of weight 2 in C, which is not the case. Therefore $g(x) = g_1(x)$ is irreducible and divides $x^n - 1$. Indeed $g(x)$ is primitive, as otherwise again there would be a code polynomial $x^p - 1$ of weight 2. □

(8.4.3) PROBLEM. *Prove that there exists a cyclic Hamming code of redundancy m and length $(q^m - 1)/(q - 1)$ over $\mathbb{F}_q$ if $\gcd((q^m - 1)/(q - 1), q - 1) = 1$. (*HINT: *For construction try as before to find such a code as a subfield subcode of a Reed-Solomon code.)*

8.4.1 Even subcodes and error detection

(8.4.4) LEMMA. *Let $F = \mathbb{F}_{2^m}$ $(m > 1)$, and let $p(x)$ be a primitive polynomial of degree m in $\mathbb{F}_2[x]$. The polynomial $g(x) = (x + 1)p(x)$ generates the even subcode E composed of all codewords of even weight in the Hamming code with generator $p(x)$. In particular, E has minimum weight 4.*

PROOF. The generator polynomial for E is a multiple of the generator polynomial $p(x)$ for the Hamming code, and so E is contained in the Hamming code. For any

$$c(x) = a(x)q(x) = a(x)(x + 1)p(x) \in E\,,$$

we have

$$c(1) = a(1)(1 + 1)p(1) = 0\,.$$

Therefore E is contained in the even subcode of the Hamming code. As the codes have the same dimension, they are equal. The Hamming code has minimum weight 3, so E has minimum weight 4. □

The even cyclic Hamming subcodes like E have often been used for detecting errors in computer applications. In that context, they are often called *CRC codes* (for 'cyclic redundancy checking'). We devote some time to detection issues for general linear and cyclic codes.

CRC codes

We recall that error detection is the particularly simple type of error control in which a received word is decoded to itself if it is a codeword and otherwise a decoding default is declared. (See Problems 2.2.2 and 2.2.3.) For a linear code, this can be gauged by whether or not the received word has syndrome $\mathbf{0}$.

(8.4.5) LEMMA. *Let C be a linear code.*
(1) *C detects any error pattern that is not a codeword.*
(2) *C detects any nonzero error pattern whose nonzero entries are restricted to the complement of an information set.*

PROOF. An error pattern that is not a codeword has nonzero syndrome as does any word in its coset.

If a codeword is 0 on an information set, then it is the $\mathbf{0}$ codeword. Thus any nonzero word that is 0 on an information set is not a codeword. □

(8.4.6) LEMMA. *A cyclic code of redundancy r detects all bursts of length at most r.*

PROOF. By Lemma 8.4.5, we must show that a codeword that is a burst of length r or less must be $\mathbf{0}$. Let $\mathbf{c}$ be such a codeword. Then it has a cyclic shift that represents a nonzero code polynomial of degree less than r. But by Theorem 8.1.1, the generator polynomial is a nonzero polynomial of minimal degree and that degree is r. Therefore $\mathbf{c} = \mathbf{0}$, as desired. □

The same argument shows that 'wrap around' burst errors, whose nonzero errors occur in a burst at the front of the word and a burst at the end, are also detected provided the combined length of the two bursts is at most r.

(8.4.7) PROBLEM. *If C is a cyclic code of redundancy r, prove that the only bursts of length $r+1$ that are codewords (and so are not detectable error patterns) are shifts of scalar multiples of the generator polynomial.*

(8.4.8) PROBLEM. *Starting with a cyclic code C of redundancy r, shorten C in its last s coordinates (or first s coordinates) by choosing all codewords that are 0 in those positions and then deleting those positions.*

*Prove that the resulting code D still can be used to detect all bursts of length at most r. (*REMARK. *The code D will no longer be cyclic and can not be relied upon for detecting burst errors that 'wrap around'.)*

(8.4.9) PROBLEM. *Have an existentialist discussion (or write such an essay) as to whether or not linear codes should be said to detect the $\mathbf{0}$ error pattern.*

Now we return to the *CRC* code E of Lemma 8.4.4, the even subcode of a binary cyclic Hamming code. E has redundancy $r = 1 + m$, where m is the redundancy of the Hamming code. Thus E can be used to detect:

(i) all odd weight errors,

(ii) all weight 2 errors,

(iii) most weight 4 errors,

(iv) all nonzero burst errors of length at most r,

(v) most burst errors of length $r + 1$.

Here C detects 'most' weight 4 errors because (at least for reasonably large r) the codewords of weight 4 form only a small fraction of the total number of words of weight 4. (See Problem 7.3.7; the total number of words of weight 4 is quartic in $n = 2^{r-1} - 1$, while the number of codewords of weight 4 is cubic in n.) The only bursts of length $r + 1$ that are codewords are the n shifts of the generator polynomial $g(x)$. (See Problem 8.4.7.) So we see that the various most likely error patterns are all detected.

EXAMPLES. (*i*) *CRC*-12 of length $2047 = 2^{11} - 1$ with generator polynomial

$$(x + 1)(x^{11} + x^2 + 1) = x^{12} + x^{11} + x^3 + x^2 + x + 1\,.$$

(*ii*) *CRC-ANSI* of length $32767 = 2^{15} - 1$ with generator polynomial

$$(x + 1)(x^{15} + x + 1) = x^{16} + x^{15} + x^2 + 1\,.$$

(*iii*) *CRC-CCITT* of length $32767 = 2^{15} - 1$ with generator polynomial

$$(x+1)(x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1) = x^{16} + x^{12} + x^5 + 1\,.$$

The last two examples have generator polynomials of the minimum weight 4. This is advantageous since the linear feedback circuitry required to implement encoding and decoding is simpler for generator polynomials of small weight.

As in Problem 8.4.8 the detection properties (*i*)-(*v*) are not lost by shortening E, so various shortened versions of even subcodes of binary cyclic Hamming codes are also used as *CRC* codes. If the code is shortened in its last s positions, then 'cyclic' encoding is still available, encoding the message polynomial $a(x)$ of degree less than $k - s$ (the dimension of the shortened code) into the code polynomial $a(x)g(x)$ of degree less than $r + k - s = n - s$ (the length of the shortened code).

8.4.2 Simplex codes and pseudo-noise sequences

As there are cyclic, binary Hamming codes of every redundancy m, there are also cyclic, binary dual Hamming codes of every dimension m. Recall that these codes are called simplex codes (or shortened first order Reed-Muller codes). They were studied previously in Section 4.3. By Theorem 8.4.2 they are precisely those cyclic, binary codes whose check polynomials are primitive.

(8.4.10) THEOREM. *Let C be a cyclic simplex code of dimension m and length $n = 2^m - 1$. Then C is composed of the codeword $\mathbf{0}$ plus the n distinct cyclic shifts of any nonzero codeword.*

PROOF. Let C have generator polynomial $g(x)$ and primitive check polynomial $h(x)$, so that $g(x)h(x) = x^n - 1$. Since $|C| = 2^m = n+1$, we need only prove that the n cyclic shifts of $g(x)$ are distinct. Suppose $g(x) = x^j g(x) \pmod{x^n - 1}$, for some $0 < j \leq n$. Thus

$$\begin{aligned} (x^j - 1)g(x) &= q(x)(x^n - 1) \\ (x^j - 1)g(x) &= q(x)g(x)h(x) \\ x^j - 1 &= q(x)h(x)\,. \end{aligned}$$

As $h(x)$ is primitive, we must have $j \geq n$ hence $j = n$. Therefore the n shifts $x^j g(x) \pmod{x^n - 1}$, for $0 \leq j < n$, are all distinct, as desired. □

(8.4.11) COROLLARY. *Let $\mathbf{0} \neq \mathbf{c} \in C$, a cyclic simplex code of dimension m. Then, for every nonzero m-tuple $\mathbf{m}$, there is exactly one set of m consecutive coordinate entries in $\mathbf{c}$ (including those that wrap around) that is equal to $\mathbf{m}$.*

PROOF. As C is cyclic, its first m coordinate positions form an information set. Every $\mathbf{m}$ occurs in these positions in exactly one codeword $\mathbf{b}$. By the theorem, $\mathbf{b}$ is a cyclic shift of $\mathbf{c}$ when $\mathbf{m}$ is one of the $2^m - 1$ nonzero m-tuples. The nonzero codeword $\mathbf{c}$ has only $n = 2^m - 1$ sets of m consecutive positions. Therefore nonzero $\mathbf{m}$ occurs exactly once among the sets of m consecutive positions in $\mathbf{c}$. □

The property described in the corollary can be thought of as a randomness property. If we were to flip an unbiased coin any number of times, then no particular combination of m consecutive heads/tails would be expected to occur more often than any other. We will call a binary sequence of length $2^m - 1$ in which each nonzero m-tuple occurs exactly once in consecutive positions a *pseudo-noise sequence* or *PN-sequence*, for short. (Here and below, when we speak of consecutive positions, we allow these positions to wrap around from the end of the word to the front.) We call it a sequence rather than word because, when we repeat it any number of times, we get a sequence of 0's and 1's whose statistical properties mimic, in part, those of a random sequence, that is, those of noise. The length $n = 2^m - 1$ is then the *period* of the sequence.

pseudo-noise sequence

PN-sequence

period

With these definitions in hand, the corollary can be restated as

(8.4.12) COROLLARY. *A nonzero codeword from a cyclic simplex code of dimension m is a PN-sequence of period $2^m - 1$.* □

There are other consequences of the *PN* definition that are similar to properties of random sequences. A *run* is a maximal set of consecutive entries consisting entirely of 0's or entirely of 1's. The length of a run is the number of its entries. In a random sequence, one would expect, for a fixed length, the same number of runs of 0's as 1's and that runs of length p would be twice as likely as runs of length $p+1$. run

(8.4.13) PROPOSITION. *Let* $\mathbf{s}$ *be a PN-sequence of period* $2^m - 1$.

(1) (Run balance) *There are exactly* 2^{m-p-2} *runs of* 0*'s of length* p $(\leq m-2)$ *and exactly* 2^{m-p-2} *runs of* 1*'s of length* p $(\leq m-2)$. *The sequence* $\mathbf{s}$ *contains exactly* 2^{m-1} *runs.*

(2) (General balance) *If* $\mathbf{p}$ *is a nonzero* p*-tuple with* $p \leq m$*, then* $\mathbf{p}$ *occurs in consecutive positions of* $\mathbf{s}$ *exactly* 2^{m-p} *times. If* $\mathbf{p}$ *is a* p*-tuple of* 0*'s, then it occurs in consecutive positions exactly* $2^{m-p} - 1$ *times. In particular,* $\mathbf{s}$ *has weight* 2^{m-1}.

PROOF. For (2), the p-tuple $\mathbf{p}$ is the initial segment of 2^{m-p} distinct m-tuples. If $\mathbf{p}$ is nonzero, then each of these m-tuples occurs within $\mathbf{s}$. If $\mathbf{p} = \mathbf{0}$, then the m-tuple $\mathbf{0}$ is the only completion of $\mathbf{p}$ that does not occur within $\mathbf{s}$. In particular, the 1-tuple 1 occurs exactly 2^{m-1} times, completing (2).

A run $aa\cdots aa$ of length p $(\leq m-2)$ corresponds to a $(p+2)$-tuple $baa\cdots aab$ with $\{a,b\} = \{0,1\}$ (which is never $\mathbf{0}$). Therefore by (2), the number of runs $aa\cdots aa$ of length p is $2^{m-(p+2)} = 2^{m-p-2}$.

If $m = 1$, then certainly there is only one run. For $m \geq 2$, a transition between two runs occurs precisely when we encounter either 01 or 10. By (2) there are 2^{m-2} of each. Therefore the number of runs, being equal to the number of transitions, is $2^{m-2} + 2^{m-2} = 2^{m-1}$. □

Although pseudo-noise and pseudo-random sequences have been studied a great deal, there is no consensus about the terminology or definitions. In some places there is no distinction made between *PN*-sequences in general and those special ones coming from simplex codes (so that Corollary 8.4.12 becomes the definition). We will call the nonzero codewords of cyclic simplex codes *m-sequences*. (This is an abbreviation for *maximal length feedback shift register sequences*.) m-sequences

It might seem more natural to consider sequences of length 2^m in which every m-tuple occurs. Such sequences are called *DeBruijn cycles*. The two concepts are in fact equivalent. If in a *PN*-sequence $\mathbf{s}$ of period $2^m - 1$ we locate the unique run of $m-1$ 0's, then we can construct a DeBruijn cycle by inserting one further 0 into the run. Conversely, if we delete a 0 from the unique run of m 0's in a DeBruijn cycle, we are left with a *PN*-sequence. Given the connection with simplex codes, we prefer the present formulation. DeBruijn cycles

(8.4.14) PROBLEM. *Prove that every PN-sequence of length* 7 *is an* m*-sequence.*

*(*REMARK. *Up to cycling, there are* 16 *PN-sequences of length* 15*, only* 2 *of which are m-sequences.)*

An important property of m-sequences is not shared by all *PN*-sequences.

(8.4.15) LEMMA. (Shift-and-add property) *If* $\mathbf{s}$ *is an m-sequence and* $\mathbf{s}'$ *is a cyclic shift of* $\mathbf{s}$*, then* $\mathbf{s}+\mathbf{s}'$ *is also a cyclic shift of* $\mathbf{s}$ *(or* $\mathbf{0}$*). In particular nonzero* $\mathbf{s}+\mathbf{s}'$ *is itself an m-sequence.*

PROOF. This is a direct consequence of Theorem 8.4.10. □

(8.4.16) PROBLEM. *Prove that a PN-sequence possessing the shift-and-add property must be an m-sequence.*

One last property is more striking if we change to the $\pm$-version of the simplex code, as described in Section 4.3. With each binary sequence $\mathbf{s}$ we associate the ±1-sequence $\mathbf{s}^*$ by replacing each 0 with the real number 1 and each 1 with the real number -1. If $\mathbf{s}$ is an m-sequence, then we abuse terminology by also referring to the associated sequence $\mathbf{s}^*$ as an m-sequence.

(8.4.17) PROPOSITION. (Perfect autocorrelation)
If $(s_0,\dots,s_{n-1})=\mathbf{s}^*\in\{\pm1\}^n\subset\mathbb{R}^n$ *is an m-sequence with* $n=2^m-1$*, then*

$$\begin{array}{rcll}\sum_{i=0}^{n-1} s_i s_{i+p} & = & n & \text{for } p=0\\ & = & -1 & \text{for } 0<p<n\,,\end{array}$$

where indices are all read modulo n.

PROOF. The summation is the dot product of $\mathbf{s}^*$ with a cyclic shift of itself. The associated binary vectors are also cyclic shifts and are either equal (when $p=0$) or at Hamming distance 2^{m-1} by Proposition 8.4.13(2) and Lemma 8.4.15. By Lemma 4.3.4 the dot product is $n\,(=2^m-1)$ when $p=0$ and otherwise $(2^m-1)-2\cdot2^{m-1}=-1$. (In fact this proposition is nearly equivalent to Lemma 4.3.5.) □

autocorrelation function

The function $a(p)=\sum_{i=0}^{n-1}s_is_{i+p}$, for $0\le p<n$, of the proposition is the *autocorrelation function* for $\mathbf{s}^*$. As n is odd, the sequences could never be orthogonal; so the proposition says, in a sense, that $\mathbf{s}^*$ and its shifts are as close to being orthogonal as possible. This is why the autocorrelation function is called perfect.

Thus the ±1 m-sequences are very unlike nontrivial shifts of themselves. For this reason, they are at times used for synchronization of signals. They are also used for modulation of distinct signals in multiple user situations. This is an example of spread spectrum communication. The idea is that multiplication by a *PN*-sequence will make a coherent signal look noise-like (taking its usual spiked frequency spectrum and spreading it out toward the flat spectrum of noise).

For such applications, it is often helpful to have not just one sequence with good autocorrelation properties but large families of them with good crosscorrelation properties. The constructions of such families may start from nice m-sequences. Their investigation is of on-going interest.

Pseudo-random binary sequences are also important for cryptography. In that context m-sequences are bad, since the shift-and-add property implies that they have low computational complexity.

Chapter 9

Weight and Distance Enumeration

The weight and distance enumerators record the weight and distance information for the code. In turn they can be analyzed to reveal properties of the code. The most important result is MacWilliams' Theorem, which we prove several times. We also prove the related Delsarte Bound and Lloyd's Theorem.

9.1 Basics

The basic definitions are:

DEFINITION. Let code $C \subseteq F^n$ (F a field) contain c_i codewords of weight i, for $i = 1, \ldots n$. Then the *weight enumerator* is

weight enumerator

$$W_C(z) = \sum_{\mathbf{c} \in C} z^{w_H(\mathbf{c})} = \sum_{i=0}^{n} c_i z^i \in \mathbb{Z}[z] \,.$$

The *homogeneous weight enumerator* is

homogeneous weight enumerator

$$W_C(x, y) = x^n W_C(y/x) = \sum_{i=0}^{n} c_i x^{n-i} y^i \in \mathbb{Z}[x, y] \,.$$

Actually these definitions make sense whenever the alphabet admits addition, an example of interest being $F = \mathbb{Z}_s$.

DEFINITION. The *distance enumerator* of the code A is given by

distance enumerator

$$W_A(z) = |A|^{-1} \sum_{\mathbf{c}, \mathbf{d} \in A} z^{d_H(\mathbf{c}, \mathbf{d})} \in \mathbb{Q}[z] \,.$$

This can be defined for any alphabet. The notation does not greatly conflict with that above, since the distance enumerator of A equals the weight enumerator

of A when A is linear. (Indeed, for a code defined over an alphabet admitting addition, we can translate each codeword to the **0**-word and get an associated weight enumerator. The distance enumerator is then the average of these weight enumerators.) One could also define a homogeneous distance enumerator.

The basic results are that of MacWilliams:

(9.1.1) THEOREM. (MACWILLIAMS' THEOREM.) *Let C be a $[n,k]$ linear code over $\mathbb{F}_s$. Set*

$$W_C(z) = \sum_{i=0}^{n} c_i\, z^i \text{ and } W_{C^\perp}(z) = \sum_{i=0}^{n} c_i^\perp z^i\,.$$

Then

(1) $W_C(z) = |C^\perp|^{-1} \sum_{i=0}^{n} c_i^\perp (1+(s-1)z)^{n-i}(1-z)^i$ *, and*

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

and its nonlinear relative due to Delsarte:

(9.1.2) THEOREM. (DELSARTE'S THEOREM.) *Let A be a code in F^n with distance enumerator $W_A(z) = \sum_{i=0}^{n} a_i z^i$. Define the rational numbers b_m by*

$$|A|^{-1} \sum_{i=0}^{n} a_i (1+(s-1)z)^{n-i}(1-z)^i = \sum_{m=0}^{n} b_m z^m\,.$$

Then $b_m \geq 0$, for all m.

These two results are related to Lloyd's Theorem 9.4.9, which states that certain polynomials associated with perfect codes must have integral roots. Lloyd's Theorem is the most powerful tool available for proving nonexistence results for perfect codes.

9.2 MacWilliams' Theorem and performance

In this section we relate weight enumerators to code performance. This leads to a first proof of MacWilliams' Theorem. For easy of exposition, we shall restrict ourselves to the consideration of binary linear codes on the $BSC(p)$ throughout this section. Let C be a binary $[n,k]$ linear code. (See Theorem 9.4.8 below for the general case of MacWilliams' Theorem 9.1.1.)

We begin with performance analysis for the binary linear code C on the $BSC(p)$ under the basic error detection algorithm $\mathbf{SS}_0 = \mathcal{D}$:

Algorithm $\mathcal{D}$:
receive $\mathbf{r}$;
if $\mathbf{r} \in C$, then decode to $\mathbf{r}$;
otherwise decode to ∞.

As before, we view the received vector $\mathbf{r}$ as being the sum of the transmitted codeword $\mathbf{c}$ and an error word $\mathbf{e}$, that is, $\mathbf{r} = \mathbf{c} + \mathbf{e}$. There are three things that can happen:

correct decoding (error vector $\mathbf{e} = \mathbf{0}$),
error detected (error vector $\mathbf{e} \notin C$),
and false decoding (error vector $\mathbf{e} \in C$, $\mathbf{e} \neq \mathbf{0}$).

The probability of correct decoding is q^n (where $q = 1 - p$). The probability of the other two events can be calculated using the weight enumerator of C. We calculate them in terms of the probability that decoding results in a guess at a codeword, whether or not that guess is correct.

(9.2.1) PROPOSITION. *Let P_D be the probability of detecting an error, P_E the probability of false decoding, and P_R the probability of getting a decoding result.*

(1) $P_R = q^n + P_E$.
(2) $P_R + P_D = 1$.
(3) $P_R = \sum_{i=0}^{n} c_i \, q^{n-i} p^i = W_C(q, p)$.

PROOF. The first two parts are clear. For the third, observe that we have a decoding result precisely when the error word is a codeword. The chance of a word of weight w occuring as an error word is $q^{n-w} p^w$. □

Next we use the dual code $C^\perp$ to calculate P_R in a different way. MacWilliams' Theorem results from equating the two probabilities. (This proof of MacWilliams' Theorem follows Chang and Wolf, 1980.)

Set $M = 2^{n-k}$, and let $C^\perp = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_j, \ldots, \mathbf{h}_M\}$. For any $\mathbf{r} \in \mathbb{F}_2^n$, we let $s_j(\mathbf{r}) = \mathbf{h}_j \cdot \mathbf{r}$ and

$$S(\mathbf{r}) = (s_1(\mathbf{r}), s_2(\mathbf{r}), \ldots, s_j(\mathbf{r}), s_M(\mathbf{r})) \in \mathbb{F}_2^M \,,$$

the "total syndrome" of $\mathbf{r}$. We have

$$\begin{array}{lllll} \mathbf{r} \text{ gives a result} & \text{iff} & \mathbf{r} \in C & \text{iff} & \mathbf{e} \in C \\ & \text{iff} & S(\mathbf{r}) = \mathbf{0} & \text{iff} & S(\mathbf{e}) = \mathbf{0} \end{array}$$

and

$$\begin{array}{lllll} \mathbf{r} \text{ gives a detected error} & \text{iff} & \mathbf{r} \notin C & \text{iff} & \mathbf{e} \notin C \\ & \text{iff} & S(\mathbf{r}) \neq \mathbf{0} & \text{iff} & S(\mathbf{e}) \neq \mathbf{0} \\ & \text{iff} & s_j(\mathbf{r}) \neq 0, \text{ some } j & \text{iff} & s_j(\mathbf{e}) \neq 0, \text{ some } j \,. \end{array}$$

Of course, for a fixed $\mathbf{e}$ and j, the probability that $S(\mathbf{e}) \neq \mathbf{0}$ or that $s_j(\mathbf{e}) \neq 0$ is either 0 or 1. Indeed

(9.2.2) LEMMA.

$$\begin{aligned} Prob(S(\mathbf{e}) \neq 0 \,|\, \mathbf{e}) &= Prob(s_j(\mathbf{e}) \neq 0, \text{ some } j \,|\, \mathbf{e}) \\ &= \frac{1}{2^{n-k-1}} \sum_{j=1}^{M} Prob(s_j(\mathbf{e}) \neq 0 \,|\, \mathbf{e}, j) \,. \end{aligned}$$

PROOF. The sum is exactly the weight of $S(\mathbf{e})$. The number of entries 0 in $S(\mathbf{e})$ is the cardinality of $\mathbf{e}^\perp \cap C^\perp$, and so is $M = 2^{n-k}$ if $\mathbf{e} \in C$ and is $M/2 = 2^{n-k-1}$ if $\mathbf{e} \notin C$. Therefore $w_H(\mathbf{S}(\mathbf{e})) \neq 0$ if and only if $w_H(\mathbf{S}(\mathbf{e})) = 2^{n-k-1}$. □

From the lemma we get

$$\begin{aligned}
P_D &= Prob(S(\mathbf{e}) \neq 0) \\
&= \sum_{\mathbf{e}\in\mathbb{F}_2^n} Prob(\mathbf{e}) Prob(S(\mathbf{e}) \neq 0 \,|\, \mathbf{e}) \\
&= \sum_{\mathbf{e}\in\mathbb{F}_2^n} Prob(\mathbf{e}) Prob(s_j(\mathbf{e}) \neq 0, \text{ some } j \,|\, \mathbf{e}) \\
&= \sum_{\mathbf{e}\in\mathbb{F}_2^n} Prob(\mathbf{e}) \frac{1}{2^{n-k-1}} \sum_{j=1}^{M} Prob(s_j(\mathbf{e}) \neq 0 \,|\, \mathbf{e}, j) \\
&= \frac{1}{2^{n-k-1}} \sum_{j=1}^{M} \sum_{\mathbf{e}\in\mathbb{F}_2^n} Prob(\mathbf{e}) Prob(s_j(\mathbf{e}) \neq 0 \,|\, \mathbf{e}, j) \\
&= \frac{1}{2^{n-k-1}} \sum_{j=1}^{M} Prob(s_j(\mathbf{e}) \neq 0 \,|\, j)
\end{aligned}$$

Therefore of interest is

(9.2.3) LEMMA. *For $w_H(\mathbf{h}_j) = w_j$,*

$$\begin{aligned}
Prob(s_j(\mathbf{e}) \neq 0 \,|\, j) &= \sum_{\substack{\text{odd} \\ i=0}}^{w_j} \binom{w_j}{i} q^{w_j - i} p^i \\
&= (1 - (q-p)^{w_j})/2 \,.
\end{aligned}$$

PROOF. Let $l_1, \ldots, l_{w_j}$ be the nonzero coordinate positions of $\mathbf{h}_j$. Then $s_j(\mathbf{e}) = \mathbf{h}_j \cdot \mathbf{e} \neq 0$ if and only if there are an odd number of 1's among the positions $\mathbf{e}_{l_i}$ for $i = 1, \ldots, w_j$. This gives the first equality. The rest follows from Lemma 9.2.4 below, as $q + p = 1$. □

(9.2.4) LEMMA. (1) $\sum_{i=0}^{w} \binom{w}{i} a^{w-i} b^i = (a+b)^w$.
(2) $\sum_{i=0}^{w} \binom{w}{i} (-1)^i a^{w-i} b^i = (a-b)^w$.
(3) $\sum_{\text{odd } i=0}^{w} \binom{w}{i} a^{w-i} b^i = ((a+b)^w - (a-b)^w)/2$. □

Lemma 9.2.3 and the previous calculation now give

$$\begin{aligned}
P_R &= 1 - P_D \\
&= 1 - \left(\frac{1}{2^{n-k-1}} \sum_{j=1}^{M} Prob(s_j(\mathbf{e}) \neq 0 \,|\, j) \right)
\end{aligned}$$

$$\begin{aligned}
&= 1 - \left(\frac{1}{2^{n-k-1}} \sum_{j=1}^{M} (1 - (q-p)^{w_j})/2 \right) \\
&= 1 - \left(\frac{1}{2^{n-k}} \sum_{j=1}^{M} (1 - (q-p)^{w_j}) \right) \\
&= 1 - \frac{1}{2^{n-k}} \sum_{j=1}^{M} 1 + \frac{1}{2^{n-k}} \sum_{j=1}^{M} (q-p)^{w_j} \\
&= \frac{1}{2^{n-k}} \sum_{j=1}^{M} (q-p)^{w_j} \\
&= \frac{1}{2^{n-k}} \sum_{i=0}^{n} c_i^{\perp} (q-p)^i ,
\end{aligned}$$

where $\sum_{i=0}^{n} c_i^{\perp} z^i = W_{C^{\perp}}(z)$.

Comparing this with Proposition 9.2.1, we obtain

(9.2.5) PROPOSITION. $\sum_{i=0}^{n} c_i \, q^{n-i} p^i = P_R = \frac{1}{2^{n-k}} \sum_{i=0}^{n} c_i^{\perp} (q-p)^i$. □

PROOF OF MACWILLIAMS' THEOREM 9.1.1 (BINARY CASE):
In the equation of the proposition, replace p by $\frac{z}{1+z}$ and $q = 1-p$ by $\frac{1}{1+z}$, to get

$$\sum_{i=0}^{n} c_i \left(\frac{1}{1+z} \right)^{n-i} \left(\frac{z}{1+z} \right)^{i} = \frac{1}{(1+z)^n} \sum_{i=0}^{n} c_i \, z^i = \frac{1}{2^{n-k}} \sum_{i=0}^{n} c_i^{\perp} \left(\frac{1-z}{1+z} \right)^{i} ,$$

hence

$$\sum_{i=0}^{n} c_i \, z^i = \frac{1}{2^{n-k}} \sum_{i=0}^{n} c_i^{\perp} (1+z)^{n-i} (1-z)^i .$$

These two polynomial functions are equal when evaluated at any $0 \le p < 1$, hence for all $z \ge 0$. We conclude that the equality is still valid in the polynomial ring $\mathbb{Q}[z]$. This gives a first proof of MacWilliams' Theorem 9.1.1 in the binary case. □

REMARK. The full version of MacWilliams' Theorem 9.1.1 for linear codes over $\mathbb{F}_s$ can be proven with exactly the same approach, evaluating in two ways the performance of error detection on the sSC(p). A proof of MacWilliams' Theorem in full generality is given below in Theorem 9.4.8(2).

Next we consider performance at the other end of the decoding spectrum—maximum likelihood decoding for error correction. The weight enumerator of a linear code can still be used to help us bound the probability of decoding falsehood $P_E = P_C(\mathbf{MLD})$.

(9.2.6) THEOREM. *When decoding the binary linear code C on the $BSC(p)$ (with $p \le \frac{1}{2}$) under* **MLD**, *we have*

$$P_C(\mathbf{MLD}) \le \sum_{w=1}^{n} c_w \, E_w \,,$$

where

$$E_w = \sum_{i=\lceil w/2 \rceil}^{w} \binom{w}{i} p^i (1-p)^{w-i} \,.$$

In particular

$$P_C(\mathbf{MLD}) \le W_C\Big(2\sqrt{p(1-p)}\Big) - 1 \,.$$

PROOF. For a given nonzero codeword $\mathbf{x}$ of weight w, E_w is the probability that the error vector $\mathbf{e}$ is at least as close to $\mathbf{x}$ as it is to $\mathbf{0}$. This must be the case if, when decoding $\mathbf{r} = \mathbf{c} + \mathbf{e}$, **MLD** incorrectly prefers $\mathbf{c} + \mathbf{x}$ to $\mathbf{c} + \mathbf{0} = \mathbf{c}$. This gives the first bound on $P_{\mathbf{MLD}}$. (It is very unlikely to be tight, since a given $\mathbf{e}$ might be closer to several codewords than it is to $\mathbf{0}$.)

As $p \le \frac{1}{2}$, we have

$$\begin{aligned} E_w = \sum_{i=\lceil w/2 \rceil}^{w} \binom{w}{i} p^i (1-p)^{w-i} &\le p^{w/2}(1-p)^{w/2} \sum_{i=\lceil w/2 \rceil}^{w} \binom{w}{i} \\ &\le p^{w/2}(1-p)^{w/2} \sum_{i=1}^{w} \binom{w}{i} \\ &= p^{w/2}(1-p)^{w/2} 2^w \\ &= \Big(2\sqrt{p(1-p)}\Big)^w \,. \end{aligned}$$

Therefore

$$P_C(\mathbf{MLD}) \le \sum_{w=1}^{n} c_w \, E_w \le \sum_{w=1}^{n} c_w \, \Big(2\sqrt{p(1-p)}\Big)^w = W_C\Big(2\sqrt{p(1-p)}\Big) - 1,$$

as desired. □

This theorem is an example of the Union Bound, and our treatment follows McEliece and Van Tilborg.

9.3 Delsarte's Theorem and bounds

We use a version of the Plotkin bound to prove a "nonlinear MacWilliams' Theorem," due originally to Delsarte. Delsarte's Theorem (9.1.2, 9.3.5, 9.4.8(1)) then leads to other important bounds.

For integers m, n, s with $0 \le m \le n$ and $s \ge 2$, define the s-ary *Krawtchouk polynomial*

Krawtchouk polynomial

$$K_m(x;n,s) = \sum_{j=0}^{m} (-1)^j \binom{x}{j}\binom{n-x}{m-j}(s-1)^{m-j}\,,$$

where, by definition,

$$\binom{x}{j} = \frac{x(x-1)\cdots(x-j+1)}{j!}\,,$$

for $x \in \mathbb{R}$. For fixed m, n, s, the Krawtchouk polynomial $K_m(x;n,s)$ has degree (at most) m in x. In particular, it is uniquely determined (using, say, Lagrange interpolation) by its values at the integers $i \in \{0,1,\ldots,n\}$. Indeed its degree in x is exactly m, since the coefficient of x^m is

$$\sum_{j=0}^{m} (-1)^j \frac{1^j}{j!}\frac{(-1)^{m-j}}{(m-j)!}(s-1)^{m-j} = \frac{(-1)^m}{m!}\sum_{j=0}^{m}\binom{m}{j}(s-1)^{m-j} = \frac{(-s)^m}{m!}\,.$$

For us, the point of introduction to these interesting polynomials is

(9.3.1) PROPOSITION. *The Krawtchouk polynomial $K_m(x;n,s)$ has degree m in x. For $i \in \{0,1,\ldots,n\}$, $K_m(i;n,s)$ is the coefficient of z^m in*

$$(1+(s-1)z)^{n-i}(1-z)^i\,.$$

PROOF. The first remark was proven above. Calculating the convolution, we see that the coefficient of z^m in this product is

$$\sum_{j=0}^{m}\left(\binom{n-i}{m-j}(s-1)^{m-j}z^{m-j}\right)\left(\binom{i}{j}(-1)^j z^j\right)\,. \qquad \square$$

(9.3.2) COROLLARY. (1) $K_0(i;n,s) = 1$.
(2) $K_1(i;n,s) = (n-i)(s-1) - i = (s-1)n - si$.
(3) $K_m(0:n,s) = (s-1)^n\binom{n}{m}$. $\square$

These could also be calculated directly.

(9.3.3) COROLLARY. *For $1 \le m \le n$ and $1 \le i \le n$, we have the recursion*

$$K_m(i;n,s) = K_m(i-1;n,s) - K_{m-1}(i-1;n,s) - (s-1)K_{m-1}(i;n,s)\,.$$

PROOF. If we evaluate the coefficient of z^m on both sides of

$$((1+(s-1)z)^{n-i}(1-z)^i)(1+(s-1)z) = ((1+(s-1)z)^{n-(i-1)}(1-z)^{i-1})(1-z)\,,$$

then we find

$$K_m(i;n,s) + (s-1)K_{m-1}(i;n,s) = K_m(i-1;n,s) - K_{m-1}(i-1;n,s)\,. \qquad \square$$

Corollary 9.3.3 gives an easy recursive method for calculating the Krawtchouk coefficients, with Corollary 9.3.2(1) and (3) providing initialization.

The proposition allows us to reformulate MacWilliams' Theorem as

(9.3.4) THEOREM. (MACWILLIAMS' THEOREM IN KRAWTCHOUK FORM.) *Let A and B be $\mathbb{F}_s$-linear codes of length n with $B = A^\perp$. Set $W_A(z) = \sum_{i=0}^n a_i\, z^i$ and $W_B(z) = \sum_{i=0}^n b_i\, z^i$. Then*

$$|A|^{-1} \sum_{i=0}^{n} K_m(i;n,s) a_i = b_m\,.$$

PROOF. Set $A = C^\perp$ and $B = C$ in MacWilliams' Theorem 9.1.1. Then b_m is the coefficient of z^m in

$$W_B(z) = |A|^{-1} \sum_{i=0}^{n} a_i (1+(s-1)z)^{n-i}(1-z)^i\,,$$

and the result follows from Proposition 9.3.1. □

We will prove Delsarte's Theorem 9.1.2 in its Krawtchouk form:

(9.3.5) THEOREM. (DELSARTE'S THEOREM IN KRAWTCHOUK FORM.) *Let A be a code of length n over an alphabet of size s, and set $W_A(z) = \sum_{i=0}^n a_i\, z^i$. Then, for $0 \le m \le n$,*

$$\sum_{i=0}^{n} K_m(i;n,s)\, a_i \ge 0\,.$$

In view of Theorem 9.3.4, Delsarte's Theorem can be thought of as a "nonlinear" version of MacWilliams' Theorem. Our proof here of Delsarte's Theorem follows Simonis and DeVroedt (1991). For linear codes A we also recover MacWilliams' Theorem (in its Krawtchouk form, Theorem 9.3.4) in the process, giving us a second proof of that result.

Let A be a code of length n with size $|A| = M$, and let $W_A(z) = \sum_{i=0}^n a_i\, z^i$ be its distance enumerator. In the next lemma s is arbitrary, but after that we will restrict our attention again to the binary case $s = 2$. As before, this restriction is only for the sake of clarity. The arguments readily extend to larger alphabets. (See Theorem 9.4.8(1) below for the general case.)

(9.3.6) LEMMA. (1) $\sum_{i=0}^n a_i = M$.

(2) $M \sum_{i=0}^n i\, a_i = \sum_{\mathbf{c},\mathbf{d}\in A} d_H(\mathbf{c},\mathbf{d})$.

PROOF. We have

$$W_A(z) = M^{-1} \sum_{\mathbf{c},\mathbf{d}\in A} z^{d_H(\mathbf{c},\mathbf{d})} = \sum_{i=0}^{n} a_i\, z^i\,;$$

so

$$W_A(1) = M^{-1} \sum_{\mathbf{c},\mathbf{d}\in A} 1^{d_H(\mathbf{c},\mathbf{d})} = \sum_{i=0}^{n} a_i\,,$$

giving (1). Similarly

$$W_A(1)' = M^{-1} \sum_{\mathbf{c},\mathbf{d}\in A} d_H(\mathbf{c},\mathbf{d})\, 1^{d_H(\mathbf{c},\mathbf{d})-1} = \sum_{i=0}^{n} i\, a_i\,,$$

giving (2). □

Again direct arguments are available. The given proof of the lemma illustrates how standard generating function methods can be used with weight and distance enumerators.

Lemma 9.3.6(1) is a strong form of the 0'th Delsarte inequality:

$$\sum_{i=0}^{n} K_0(i;n,s)\, a_i = \sum_{i=0}^{n} 1 \cdot a_i = M \geq 0\,.$$

For linear A, this could be phrased as $\sum_{i=0}^{n} K_0(i;n,s)\, a_i = Mb_0$, where $b_0 = 1$ is the number of words of weight 0 in $A^\perp$.

At this point, we assume additionally that A is a binary code.

(9.3.7) LEMMA. (FIRST BINARY DELSARTE INEQUALITY.) *We have*

$$\sum_{i=0}^{n} K_1(i;n,2)\, a_i = \sum_{i=0}^{n} (n-2i) a_i \geq 0\,.$$

Indeed, if A is binary and linear, then this sum is Mb_1, where b_1 is the number of words of weight 1 in $A^\perp$.

PROOF. Let G be an $M \times n$ matrix whose rows are the codewords of A, and let w_j be the weight of the j'th column of G. In Lemma 9.3.6

$$M \sum_{i=0}^{n} i\, a_i = \sum_{\mathbf{c},\mathbf{d}\in A} d_H(\mathbf{c},\mathbf{d})$$

effectively counts pairwise distances in A by examining G row-by-row. To count instead by columns, observe that in column j a 0 of row $\mathbf{x}$ and 1 of row $\mathbf{y}$ contribute twice to the sum, once for each of $d_H(\mathbf{x},\mathbf{y})$ and $d_H(\mathbf{y},\mathbf{x})$. Thus

$$\sum_{\mathbf{c},\mathbf{d}\in A} d_H(\mathbf{c},\mathbf{d}) = 2 \sum_{j=1}^{n} w_j (M - w_j)\,.$$

Therefore

$$\begin{aligned} \sum_{i=0}^{n} i\, a_i &= 2M^{-1} \sum_{j=1}^{n} w_j (M - w_j) \\ &\leq 2M^{-1} \sum_{j=1}^{n} \frac{M^2}{4} \end{aligned}$$

$$\begin{aligned} &= \frac{n}{2}M \\ &= \frac{n}{2}\sum_{i=0}^{n} a_i\,, \end{aligned}$$

and so

$$\begin{aligned} 0 &\le 2\left(\frac{n}{2}\sum_{i=0}^{n} a_i - \sum_{i=0}^{n} i\,a_i\right) \\ &= \sum_{i=0}^{n}(n-2i)a_i\,. \end{aligned}$$

This proves the first Delsarte inequality.

If A is linear, then the various w_j are either 0 or $M/2$. Indeed w_j is 0 only when there is in $A^\perp$ a word of weight 1 whose nonzero entry is at position j, the number of such positions being b_1. Therefore

$$\begin{aligned} \sum_{i=0}^{n} i\,a_i &= 2M^{-1}\sum_{j=1}^{n} w_j(M-w_j) \\ &= (n-b_1)M/2 \end{aligned}$$

and

$$\sum_{i=0}^{n}(n-2i)a_i = nM-(n-b_1)M = b_1M\,. \qquad \square$$

(9.3.8) Corollary. (Binary Plotkin bound.) *If A is a binary code with minimum distance d and length $n < 2d$, then*

$$|A| \le \frac{2d}{2d-n}\,.$$

Proof. The first Delsarte inequality yields

$$\begin{aligned} 0 &\le \sum_{i=0}^{n}(n-2i)a_i \\ &= n+\sum_{i=d}^{n}(n-2i)a_i \\ &\le n+(n-2d)\sum_{i=d}^{n} a_i \\ &= n+(n-2d)(|A|-1) \\ &= n-(n-2d)+(n-2d)|A| \\ &= 2d+(n-2d)|A|\,. \end{aligned}$$

This implies $(2d-n)|A| \le 2d$ and so proves the Plotkin bound. $\square$

In Lemmas 9.3.6(1) and 9.3.7 we have the Delsarte inequalities for $m = 0, 1$ (and the corresponding linear interpretation). We next attack the m'th Delsarte inequality.

For a fixed m, consider a new code $A^{[m]} = \{ \mathbf{c}^{[m]} \mid \mathbf{c} \in C \}$ of length $N = \binom{n}{m}$. Starting with a codeword $\mathbf{c} \in A$, we construct the codeword $\mathbf{c}^{[m]} \in A^{[m]}$, whose entries $c_J^{[m]}$ are indexed by the m-subsets of $\{1, \ldots, n\}$, and are given by

$$c_J^{[m]} = \sum_{j \in J} c_j \,,$$

for each m-subset J.

(9.3.9) LEMMA.
(1) *If* $\mathbf{x} + \mathbf{y} = \mathbf{z}$, *then* $\mathbf{x}^{[m]} + \mathbf{y}^{[m]} = \mathbf{z}^{[m]}$.
(2) *If* $w_H(\mathbf{x}) = i$, *then* $w_H(\mathbf{x}^{[\mathbf{m}]}) = \sum_{j \text{ odd}} \binom{i}{j}\binom{n-i}{m-j}$.

PROOF. The first part is immediate. For the second part, let I be the subset of $\{1, \ldots, n\}$ whose positions hold the 1's of $\mathbf{x}$. Then the entry $x_J^{[m]}$ is 0 or 1 as $|I \cap J| = j$ is even or odd. For a fixed j, there are $\binom{i}{j}$ choices for $I \cap J$ and $\binom{n-i}{m-j}$ ways of completing this choice to an appropriate m-subset of $\{1, \ldots, n\}$. □

A particular consequence of Lemma 9.3.9 is that $A^{[m]}$ has the same number of codewords as A. The weight in (2) depends only on the original weight i, so we can define

$$w^{[m]}(i) = \sum_{\substack{\text{odd} \\ j=0}}^{m} \binom{i}{j}\binom{n-i}{m-j} .$$

If $d_H(\mathbf{x}, \mathbf{y}) = i$, then $d_H(\mathbf{x}^{[\mathbf{m}]}, \mathbf{y}^{[\mathbf{m}]}) = w^{[m]}(i)$. Therefore, for $W_{A^{[m]}}(z) = \sum_r a_r^{[m]} z^r$, we have

$$a_r^{[m]} = \sum_{w^{[m]}(i) = r} a_i \,.$$

The definition of $w^{[m]}(i)$ is to be compared with the well-known binomial identity

$$\binom{n}{m} = \sum_{j=0}^{m} \binom{i}{j}\binom{n-i}{m-j} ,$$

proved by counting m-subsets of a two-colored n set according to how many elements of each color have been selected.

PROOF OF DELSARTE'S THEOREM 9.3.5 (BINARY CASE):
By the first Delsarte inequality for $A^{[m]}$, we have

$$0 \;\leq\; \sum_{r=0}^{N} (N - 2r) a_r^{[m]}$$

$$
\begin{aligned}
&= \sum_{r=0}^{N}(N-2r)\sum_{w^{[m]}(i)=r} a_i \\
&= \sum_{r=0}^{N}\sum_{w^{[m]}(i)=r}(N-2w^{[m]}(i))a_i \\
&= \sum_{i=0}^{n}\left(\binom{n}{m}-2w^{[m]}(i)\right)a_i \\
&= \sum_{i=0}^{n}\left(\left(\sum_{j=0}^{m}\binom{i}{j}\binom{n-i}{m-j}\right)-2\left(\sum_{\substack{j=0\\ \text{odd}}}^{m}\binom{i}{j}\binom{n-i}{m-j}\right)\right)a_i \\
&= \sum_{i=0}^{n}\left(\sum_{j=0}^{m}(-1)^j\binom{i}{j}\binom{n-i}{m-j}\right)a_i \\
&= \sum_{i=0}^{n}K_m(i;n,2)\,a_i\,. \qquad \square
\end{aligned}
$$

In the case that A is linear, $A^{[m]}$ is also linear by Lemma 9.3.9(1). The sum counts $|A^{[m]}| = |A|$ times the number of weight 1 words in ${A^{[m]}}^{\perp}$. Let $\mathbf{x}$ be a word of weight 1 in $\mathbb{F}_2^N$, and suppose its unique nonzero entry is in position J, where J is an m-subset of $\{1,\ldots,n\}$. Then $\mathbf{x}$ will be in ${A^{[m]}}^{\perp}$ when all codewords of $A^{[m]}$ have J-entry 0. This happens when every codeword $\mathbf{c}$ of A has an even number of 1's in the positions of J. That is, when the word of $\mathbb{F}_2^n$ with 1's in the positions of J belongs to $A^{\perp}$. Therefore words of weight 1 in ${A^{[m]}}^{\perp}$ correspond exactly to words of weight m in $A^{\perp}$, and we have recovered MacWilliams' Theorem in its Krawtchouk form 9.3.4.

(9.3.10) THEOREM. (LINEAR PROGRAMMING BOUND.) *Let A be a code of length n over an alphabet of size s with $d_{min}(A) \geq d$. Then*

$$
\begin{aligned}
|A| \;\leq\; \max\Bigg\{ \sum_{i=0}^{n} A_i \;\Bigg|\; & A_0 = 1,\ A_i = 0,\ 1 \leq i \leq d, \\
& A_m \geq 0\,,\ \sum_{i=0}^{n} A_i K_m(i;n,s) \geq 0,\ 1 \leq m \leq n \Bigg\}\,.
\end{aligned}
$$

If $s = 2$ and d is even, then we can also assume that $A_i = 0$, for all odd i.

PROOF. For $W_A(z) = \sum_{i=0}^{n} a_i\, z^i$, the choice $A_i = a_i$ solves all the inequalities by Delsarte's Theorem 9.3.5. It has $\sum_{i=0}^{n} A_i = |A|$, by Lemma 9.3.6(1).

If $s = 2$ and d is even, then when we first puncture and then extend A, the resulting code A^* (even in the nonlinear case) has $|A^*| = |A|$ and $d_{min}(A^*) \geq d$.

Furthermore, the coefficients a_i^* of $W_{A^*}(z)$ satisfy the same inequalities as the a_i and additionally have $a_i^* = 0$, for odd i. □

As our proof of the Plotkin bound in Corollary 9.3.8 suggests, these methods can be used to find general bounds; but new bounds of this type are very difficult to prove. On the other hand, the linear programming bound is remarkably effective in specific cases, as the following example suggests.

EXAMPLE. Let C be a binary linear code of length 8 with $d_{min}(C) \geq 4$. We prove that $|C| \leq 16$ (the extended Hamming code providing an example that has exactly 16 codewords).

We have $A_0 = 1$, $A_2 = A_3 = A_5 = A_7 = 0$, and also $A_4 \geq 0$, $A_6 \geq 0$, and $A_8 \geq 0$. The Delsarte inequalities for m and $8 - m$ are equal under these circumstances, so only $m = 1, 2, 3, 4$ can be of help. In fact, those for $m = 1$ and $m = 2$ are all we need. We have (using Corollaries 9.3.2 and 9.3.3)

$$\begin{aligned} 0 &\leq \sum_{i=0}^{8} A_i K_1(i; n, s) = 8 + 0A_4 - 4A_6 - 8A_8\,; \\ 0 &\leq \sum_{i=0}^{8} A_i K_2(i; n, s) = 28 - 4A_4 + 4A_6 + 28A_8\,. \end{aligned}$$

The first inequality gives

$$A_6 \leq 2 - 2A_8,$$

so that in particular $A_8 \leq 1$. Adding the two inequalities produces

$$A_4 \leq 9 + 5A_8\,.$$

Therefore

$$\begin{aligned} |C| &\leq \sum_{i=0}^{8} A_8 \\ &= A_0 + A_4 + A_6 + A_8 \\ &\leq 1 + (9 + 5A_8) + (2 - 2A_8) + A_8 \\ &= 12 + 4A_8 \\ &\leq 16\,, \end{aligned}$$

as claimed. Indeed, in order for the sum to be 16 we must have $A_8 = 1$, in which case $0 \leq A_6 \leq 2 - 2A_8$ yields $A_6 = 0$. Also $A_4 \leq 9 + 5A_8 = 14$. As

$$\begin{aligned} \sum_{i=0}^{8} A_8 &= A_0 + A_4 + A_6 + A_8 \\ &\leq 1 + 14 + 0 + 1 \\ &\leq 16\,, \end{aligned}$$

there is a unique solution to the linear programming problem, namely

$$A_0 = 1,\ A_1 = A_2 = A_3 = A_5 = A_6 = A_7 = 0,\ A_4 = 14,\ A_8 = 1\,.$$

This corresponds to the weight enumerator $1 + 14z^4 + z^8$ for the extended binary Hamming code of length 8.

Of course this toy example could also be handled by more combinatorial methods, but the linear programming approach has better scaling properties. For instance, codes with lengths in the teens can still be handled very easily, while combinatorial approaches can already at that point require extensive case-by-case analysis.

9.4 Lloyd's theorem and perfect codes

We present MacWilliams' theorem a third time, Delsarte's theorem a second time, and Lloyd's theorem a first time.

In this section, we will be concerned with codes defined over a finite alphabet F that has the structure of a commutative ring with a multiplicative identity 1. Our main examples are fields $\mathbb{F}_s$ and rings of modular integers $\mathbb{Z}_s = \mathbb{Z} \pmod{s}$. It is important to realize that any code over a finite alphabet of s letters can be viewed as a code over $\mathbb{Z}_s$ (merely by assigning each letter a unique value from $\mathbb{Z}_s$). In particular, our proof here of Delsarte's Theorem in Theorem 9.4.8(1) does not suffer any loss of generality by restricting attention to codes over $\mathbb{Z}_s$.

In this situation we have an additive structure on F^n with identity element $\mathbf{0}$, and we have natural scalar multiplication given by

$$r(a_1, \ldots, a_j, \ldots, a_n) = (ra_1, \ldots, ra_j, \ldots, ra_n)\,,$$

for arbitrary $r \in F$. Therefore we can still talk about $r\mathbf{a} + t\mathbf{b}$, $\mathbf{c} \cdot \mathbf{d}$, $w_H(\mathbf{e})$, and so forth.

F-linear code

An *F-linear code* of length n will be, as before, any nonempty subset C of F^n that is closed under addition and scalar multiplication. The code $C^\perp$ dual to C is also defined as before:

$$C^\perp = \{\, \mathbf{v} \in F^n \mid \mathbf{v} \cdot \mathbf{c} = 0, \quad \text{for all} \quad \mathbf{c} \in C \,\}\,,$$

and is linear even if C is not.

linear character

A *linear character* χ of $(F, +)$ is a map $\chi\colon F \longrightarrow \mathbb{C}^*$ with

$$\chi(a + b) = \chi(a)\chi(b) \quad \text{for all} \quad a, b \in F\,.$$

For finite F the image of χ will be in the roots of unity, and we must have

$$\chi(0) = 1 \quad \text{and} \quad \chi(-a) = \chi(a)^{-1} = \overline{\chi(a)}\,.$$

A basic example is the trivial character $1_F(a) = 1$, for all $a \in F$. Later we will make a specific choice for χ, but for the moment χ can be any linear character of $(F, +)$.

We next define, for $\mathbf{u}, \mathbf{v} \in V = F^n$, the related notation

$$\chi(\mathbf{u}|\mathbf{v}) = \chi(\mathbf{u} \cdot \mathbf{v}) = \chi\left(\sum_{i=1}^{n} u_i v_i\right) = \prod_{i=1}^{n} \chi(u_i v_i)\,.$$

For $n = 1$, $\chi(u|v) = \chi(uv)$; and the first two parts of the next lemma are consequences of the commutativity of F, while the third part is just a restatement of the defining property for a character. The general case follows directly.

(9.4.1) LEMMA. (1) $\chi(\mathbf{u}|\mathbf{v}) = \chi(\mathbf{v}|\mathbf{u})$;
(2) *for* $a \in F$, $\chi(\mathbf{u}|a\mathbf{v}) = \chi(a\mathbf{u}|\mathbf{v}) = \chi(a|\mathbf{u}\cdot\mathbf{v})$;
(3) $\chi(\mathbf{a}+\mathbf{b}|\mathbf{v}) = \chi(\mathbf{a}|\mathbf{v}) + \chi(\mathbf{b}|\mathbf{v})$. □

We thus see that $\chi(\cdot|\cdot)$ is symmetric and biadditive on F^n.

More generally, for subsets A, B of V, we define

$$\chi(A|B) = \sum_{\mathbf{a}\in A, \mathbf{b}\in B} \chi(\mathbf{a}|\mathbf{b})\,.$$

We have before encountered the notation

$$A + B = \{\, \mathbf{a}+\mathbf{b} \mid \mathbf{a}\in A\,, \mathbf{b}\in B\,\}\,,$$

and we further write $A \oplus B$ for $A + B$ if every element of $A + B$ has a unique expression as $\mathbf{a}+\mathbf{b}$, for $\mathbf{a}\in A$ and $\mathbf{b}\in B$.

The defining property of a character χ and biadditivity then extend to

(9.4.2) LEMMA. $\chi(A\oplus B|\mathbf{v}) = \chi(A|\mathbf{v})\chi(B|\mathbf{v})$

PROOF.

$$\begin{aligned}
\chi(A\oplus B|\mathbf{v}) &= \sum_{\mathbf{a}\in A,\mathbf{b}\in B} \chi(\mathbf{a}+\mathbf{b}|\mathbf{v}) \\
&= \sum_{\mathbf{a}\in A,\mathbf{b}\in B} \chi(\mathbf{a}|\mathbf{v})\chi(\mathbf{b}|\mathbf{v}) \\
&= \sum_{\mathbf{a}\in A} \chi(\mathbf{a}|\mathbf{v}) \sum_{\mathbf{b}\in B} \chi(\mathbf{b}|\mathbf{v}) \\
&= \chi(A|\mathbf{v})\chi(B|\mathbf{v}) \qquad \square
\end{aligned}$$

REMARK. The lemma and proof remain valid for all A and B if we view $A+B$ as a multiset, keeping track of the number of different ways each element can be written as $\mathbf{a}+\mathbf{b}$. This is the "group algebra" approach, which can be very effective.

The next two lemmas are elementary but fundamental for what we are doing in this section.

(9.4.3) LEMMA. *Consider the property:*

> *(ND) F is a commutative ring with identity, and $(F,+)$ possesses a linear character χ such that, for each $0 \neq v \in F$, there is an $a_v \in F$ with $\chi(a_v v) \neq 1$.*

Then $\mathbb{F}_s$ and $\mathbb{Z}_s$ both have the property (ND).

PROOF. For $F = \mathbb{Z}_s$, let ζ be a primitive s'th root of 1 in $\mathbb{C}$. (That is, $\zeta^s = 1$ but $\zeta^i \neq 1$, for $0 < i < s$.) Then $\chi(i) = \zeta^i$ has the desired properties with respect to $a_v = 1$, for all $v \neq 0$.

Let $F = \mathbb{F}_s$ with $s = p^d$, a power of the prime p. In fact, every nontrivial linear character χ has the desired property. We give a concrete construction. Let ζ be a primitive p'th root of 1. Realize F as $\mathbb{F}_p[x] \pmod{m(x)}$ for an irreducible polynomial $m(x)$ of degree d in $\mathbb{F}_p[x]$. Each element of F is represented by a unique polynomial $f(x) \in \mathbb{F}_p[x]$ of degree less than d. Then $\chi(f(x)) = \zeta^{f(0)}$ has the desired properties. (Each $f(0)$ is in $\mathbb{F}_p = \mathbb{Z}_p$ and can be thought of as an integer.) For each $v \neq 0$, we can choose $a_v = v^{-1}$. □

If we view $\chi(\cdot|\cdot)$ as a symmetric, biadditive form on F^n, then Property (ND) of the lemma says that, at least for F^1, the form is nondegenerate:

$$0 = \{\, v \in F^1 \mid \chi(a|v) = 1, \text{ for all } \ a \in F^1 \,\}.$$

The next lemma continues this line of thought.

From now on we will assume that the alphabet F has a character χ of $(F, +)$ satisfying Property (ND) of Lemma 9.4.3. We choose and fix such a character χ. Our main examples remain $\mathbb{Z}_s$ and $\mathbb{F}_s$.

(9.4.4) LEMMA. *Let $\mathbf{v} \in V$.*

(1) *Always*

$$\begin{aligned} \chi(V|\mathbf{v}) &= |V| \ \text{ if } \ \mathbf{v} = 0 \\ &= \ 0 \ \text{ otherwise.} \end{aligned}$$

(2) *If W is an F-linear code in V, then*

$$\begin{aligned} \chi(W|\mathbf{v}) &= |W| \ \text{ if } \ \mathbf{v} \in W^\perp \\ &= \ 0 \ \text{ otherwise.} \end{aligned}$$

PROOF. If $\mathbf{0} \neq \mathbf{v} \in V$, then by Property (ND) of Lemma 9.4.3 there is a word $\mathbf{a} \in V$ with weight 1 and $\mathbf{a} \cdot \mathbf{v} \neq 0$. Therefore $V^\perp = \{\mathbf{0}\}$, and (1) is a special case of (2).

For (2), if $\mathbf{v} \in W^\perp$, then

$$\chi(W|\mathbf{v}) = \sum_{\mathbf{w} \in W} 1 = |W|\,.$$

Therefore to complete (2) and the lemma we may assume that there is a $\mathbf{w} \in W$ with $v = \mathbf{w} \cdot \mathbf{v} \neq 0$.

By Property (ND) of Lemma 9.4.3, there is an $a = a_v \in F$ with $\chi(av) \neq 1$. Therefore, for $\mathbf{a} = a\mathbf{w}$,

$$\chi(\mathbf{a}|\mathbf{v}) = \chi(a|\mathbf{w} \cdot \mathbf{v}) = \chi(a|v) \neq 1\,,$$

by Lemma 9.4.1(2).

Now we have

$$\begin{aligned}\chi(W|\mathbf{v}) &= \chi(W \oplus \mathbf{a}|\mathbf{v}) \\ &= \chi(W|\mathbf{v})\chi(\mathbf{a}|\mathbf{v})\end{aligned}$$

by Lemma 9.4.2. Therefore

$$0 = \chi(W|\mathbf{v})(\chi(\mathbf{a}|\mathbf{v}) - 1)\,,$$

and $\chi(W|\mathbf{v}) = 0$, as required. □

(9.4.5) COROLLARY. *Suppose that, for some set of constants $\alpha_{\mathbf{u}}$,*

$$\sum_{\mathbf{u}\in V} \alpha_{\mathbf{u}}\,\chi(\mathbf{u}|\mathbf{v}) = 0\,,$$

for all $\mathbf{0} \neq \mathbf{v} \in V$. Then $\alpha_{\mathbf{u}} = \alpha$ is constant, for all $\mathbf{u} \in V$.

PROOF. By Lemma 9.4.4(1), a constant choice $\alpha_{\mathbf{u}} = \alpha$ does have the stated property. In particular, after subtracting an appropriate constant from each coefficient, we could assume that $\sum_{\mathbf{u}\in V} \alpha_{\mathbf{u}}\,\chi(\mathbf{u}|\mathbf{v}) = 0$ holds for all $\mathbf{v}$, including $\mathbf{0}$. We do so, and then aim to prove that each $\alpha_{\mathbf{u}}$ equals 0.

For fixed but arbitrary $\mathbf{z} \in V$, we have

$$\begin{aligned}0 &= \sum_{\mathbf{v}\in V} 0 \cdot \overline{\chi(\mathbf{z}|\mathbf{v})} \\ &= \sum_{\mathbf{v}\in V} \left(\sum_{\mathbf{u}\in V} \alpha_{\mathbf{u}}\,\chi(\mathbf{u}|\mathbf{v})\right) \overline{\chi(\mathbf{z}|\mathbf{v})} \\ &= \sum_{\mathbf{u}\in V} \alpha_{\mathbf{u}} \left(\sum_{\mathbf{v}\in V} \chi(\mathbf{u}|\mathbf{v})\overline{\chi(\mathbf{z}|\mathbf{v})}\right) \\ &= \sum_{\mathbf{u}\in V} \alpha_{\mathbf{u}} \sum_{\mathbf{v}\in V} \chi(\mathbf{u}-\mathbf{z}|\mathbf{v}) \\ &= \alpha_{\mathbf{z}}|V|\end{aligned}$$

by Lemma 9.4.4(1). □

(9.4.6) PROPOSITION. *For all $\mathbf{v} \in V$ with $w_H(\mathbf{v}) = i$,*

$$\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{v}) = (1 + (s-1)z)^{n-i}(1-z)^i\,.$$

PROOF. For all $(v_1, \ldots, v_n) = \mathbf{v} \in V$, we have

$$\begin{aligned}\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{v}) &= \sum_{\mathbf{u}\in V} \prod_{j=1}^{n} z^{w_H(u_j)}\chi(u_j|v_j) \\ &= \prod_{j=1}^{n} \sum_{u\in F} z^{w_H(u)}\chi(u|v_j)\end{aligned}$$

by distributivity. Here

$$\sum_{u\in F} z^{w_H(u)}\chi(u|v_j) = 1 + z\,\chi(F\setminus\{0\}\,|v_j)$$

which, by the case $n=1$ of Lemma 9.4.4(1), is $1+(s-1)z$ when $v_j=0$ and is $1-z$ when $v_j\neq 0$. Therefore

$$\begin{aligned}\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{v}) &= \prod_{j=1}^{n}\sum_{u\in F} z^{w_H(u)}\chi(u|v_j)\\ &= (1+(s-1)z)^{n-w_H(\mathbf{v})}(1-z)^{w_H(\mathbf{v})}\,,\end{aligned}$$

as claimed. □

Let $Y_m = \{\,\mathbf{x}\in V\,|\,w_H(\mathbf{x})=m\,\}$, so that the sphere of radius e centered at $\mathbf{0}$, $S_e = S_e(\mathbf{0})$, is the disjoint union of the Y_m, for $m=1,\dots,e$.

(9.4.7) COROLLARY. *Let $w_H(\mathbf{v})=i$.*
(1) $\chi(Y_m|\mathbf{v}) = K_m(i;n,s)$,
(2) $\chi(S_e|\mathbf{v}) = \sum_{m=0}^{e} K_m(i;n,s)$.

PROOF. By the proposition, $\chi(Y_m|\mathbf{v})$ is the coefficient of z^m in

$$(1+(s-1)z)^{n-i}(1-z)^i\,.$$

By Proposition 9.3.1 this coefficient is also $K_m(i;n,s)$. This gives (1), and (2) follows directly. □

(9.4.8) THEOREM. *Let A be a code in F^n with distance enumerator $W_A(z) = \sum_{i=0}^{n} a_i z^i$. Define the rational numbers b_m by*

$$|A|^{-1}\sum_{i=0}^{n} a_i(1+(s-1)z)^{n-i}(1-z)^i = \sum_{m=0}^{n} b_m z^m\,.$$

Then
(1) (DELSARTE'S THEOREM 9.1.2.) *$b_m \geq 0$, for all m. (Indeed we have $b_m = |A|^{-2}\sum_{w_H(\mathbf{u})=m}|\chi(\mathbf{u}|A)|^2$.)*
(2) (MACWILLIAMS' THEOREM 9.1.1.) *If A is an F-linear code, then $W_{A^\perp}(z) = \sum_{m=0}^{n} b_m z^m$.*

PROOF. We calculate

$$\sum_{\mathbf{c},\mathbf{d}\in A}\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{c}-\mathbf{d})$$

in two different ways.

By Proposition 9.4.6,

$$\begin{aligned}\sum_{\mathbf{c},\mathbf{d}\in A}\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{c}-\mathbf{d}) &= \sum_{\mathbf{c},\mathbf{d}\in A}(1+(s-1)z)^{n-w_H(\mathbf{c}-\mathbf{d})}(1+z)^{w_H(\mathbf{c}-\mathbf{d})}\\ &= |A|\sum_{i=0}^{n} a_i(1+(s-1)z)^{n-i}(1+z)^i\,,\end{aligned}$$

which is $|A|^2$ times the lefthand side of the definition.

On the other hand

$$\begin{aligned}\sum_{\mathbf{c},\mathbf{d}\in A}\sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|\mathbf{c}-\mathbf{d}) &= \sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\sum_{\mathbf{c},\mathbf{d}\in A}\chi(\mathbf{u}|\mathbf{c}-\mathbf{d})\\ &= \sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\sum_{\mathbf{c},\mathbf{d}\in A}\chi(\mathbf{u}|\mathbf{c})\chi(\mathbf{u}|-\mathbf{d})\\ &= \sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|A)\chi(\mathbf{u}|-A)\\ &= \sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}\chi(\mathbf{u}|A)\overline{\chi(\mathbf{u}|A)}\\ &= \sum_{\mathbf{u}\in V} z^{w_H(\mathbf{u})}|\chi(\mathbf{u}|A)|^2\\ &= \sum_{m=0}^{n}\left(\sum_{w_H(\mathbf{u})=m}|\chi(\mathbf{u}|A)|^2\right)z^m\,.\end{aligned}$$

We conclude that

$$|A|\sum_{i=0}^{n} a_i(1+(s-1)z)^{n-i}(1+z)^i = \sum_{m=0}^{n}\left(\sum_{w_H(\mathbf{u})=m}|\chi(\mathbf{u}|A)|^2\right)z^m\,.$$

Therefore

$$b_m = |A|^{-2}\sum_{w_H(\mathbf{u})=m}|\chi(\mathbf{u}|A)|^2 \geq 0\,,$$

proving Delsarte's Theorem.

Furthermore, if A is linear, then by Lemma 9.4.4(2)

$$\begin{aligned}\chi(\mathbf{u}|A) &= |A| \text{ if } \mathbf{u}\in A^\perp\\ &= 0 \text{ otherwise.}\end{aligned}$$

Therefore

$$\begin{aligned}b_m &= |A|^{-2}\sum_{w_H(\mathbf{u})=m}|\chi(\mathbf{u}|A)|^2\\ &= |A|^{-2}\sum_{\substack{w_H(\mathbf{u})=m\\ \mathbf{u}\in A^\perp}}|A|^2\\ &= |\{\,\mathbf{u}\,|\,w_H(\mathbf{u})=m,\mathbf{u}\in A^\perp\,\}|\,.\end{aligned}$$

proving MacWilliams' Theorem. □

This proof of MacWilliams' Theorem is essentially one of the two given in the original paper (and thesis) of MacWilliams for linear codes over fields. Its modification to prove Delsarte's Theorem as well is due to Welch, McEliece, and Rumsey (1974).

Here we have proved MacWilliams' Theorem for a more general class of codes than the linear codes of Theorem 9.1.1, namely for those codes that are F-linear, where F has Property (ND) of Lemma 9.4.3. Many properties of linear codes over fields go over to these codes. For instance, substituting 1 into the equations of Theorem 9.4.8, we learn that, for the F-linear code A,

$$|A|^{-1}\, s^n = |A|^{-1} \sum_{i=0}^{n} a_i (1 + (s-1)1)^{n-i} (1-1)^i = \sum_{m=0}^{n} b_m 1^m = |A^\perp|\,.$$

That is, $|A||A^\perp| = |V|$, whence $A^{\perp\perp} = A$ (things that could also be proved directly).

(9.4.9) THEOREM. (LLOYD'S THEOREM.) *Let $C \subseteq F^n$ be a perfect e-error-correcting code. Then the polynomial*

$$\Psi_e(x) = \sum_{i=0}^{e} K_i(x; n, s)$$

of degree e has e distinct integral roots in $\{1, \ldots, n\}$.

PROOF. The basic observation is that, since C is a perfect e-error-correcting code,

$$S_e \oplus C = V\,.$$

Therefore, by Lemma 9.4.2

$$\chi(S_e|\mathbf{v})\, \chi(C|\mathbf{v}) = \chi(V|\mathbf{v})\,,$$

for all $\mathbf{v} \in V$. As $V = S_n$, we have by Corollary 9.4.7

$$\Psi_e(x)\, \chi(C|\mathbf{v}) = \Psi_n(x)\,,$$

where $x = w_H(\mathbf{v})$ and $\Psi_j(x) = \sum_{i=0}^{j} K_i(x; n, s)$.

By Proposition 9.3.1, $K_i(x; n, s)$ is a polynomial of degree i in x, so $\Psi_j(x)$ has degree j in x. In particular, $\Psi_n(x) = \chi(V|\mathbf{v})$ has degree n. But by Lemma 9.4.4(1) it has roots $x = 1, \ldots, n$. Therefore

$$\Psi_n(x) = c(x-1)(x-2)\cdots(x-n)\,,$$

for an appropriate constant c (which can be calculated using Corollary 9.3.2). We conclude that

$$\Psi_e(x)\, \chi(C|\mathbf{v}) = c(x-1)(x-2)\cdots(x-n)\,,$$

for $x = w_H(\mathbf{v})$.

As the polynomial $\Psi_e(x)$ has degree e in x, the theorem will be proven once we can show that, for at least e values of $m \neq 0$, there are words $\mathbf{v} \in Y_m$ with $\chi(C|\mathbf{v}) \neq 0$. By Theorem 9.4.8(1), this is equivalent to proving that $|\{\, m \neq 0 \,|\, b_m \neq 0 \,\}| \geq e$. But this is immediate from Proposition 9.4.10 below. □

(9.4.10) PROPOSITION. *For the e-error-correcting code A with the b_m defined as in Theorem 9.4.8, we have*

$$|\{\, m \neq 0 \,|\, b_m \neq 0 \,\}| \geq e\,.$$

PROOF. Let $N(A) = \{\, m \neq 0 \,|\, b_m \neq 0 \,\}$ and $g = |N(A)|$, and assume that $g \leq e$. Define the polynomial $a(x) = \prod_{m \in N(A)}(x - m)$, of degree g (an empty product being taken as 1). Therefore $a(m) = 0$ when $m \in N(A)$, whereas $\chi(A|\mathbf{v}) = 0$ whenever $0 \neq m = w_H(\mathbf{v}) \notin N(A)$, by Theorem 9.4.8(1).

As each $K_i(x; n, s)$ has degree i in x (by Proposition 9.3.1), there are constants α_i (not all 0) with

$$a(x) = \sum_{i=0}^{g} \alpha_i\, K_i(x; n, s)\,.$$

Using Corollary 9.4.7(1), we get, for all $\mathbf{v} \neq \mathbf{0}$,

$$\begin{aligned} 0 &= a(w_H(\mathbf{v}))\chi(A|\mathbf{v}) \\ &= \left(\sum_{i=0}^{g} \alpha_i\, \chi(Y_i|\mathbf{v})\right) \chi(A|\mathbf{v}) \\ &= \sum_{i=0}^{g} \alpha_i\, \chi(Y_i \oplus A|\mathbf{v}) \\ &= \sum_{i=0}^{g} \sum_{\mathbf{y} \in Y_i, \mathbf{a} \in A} \alpha_i\, \chi(\mathbf{y} + \mathbf{a}|\mathbf{v})\,. \end{aligned}$$

From Corollary 9.4.5 we learn $\alpha_i = \alpha$ is a nonzero constant function of i and that every element $\mathbf{u} \in V$ is equal to some $\mathbf{y} + \mathbf{a}$. In particular, it must be possible to write the word $\mathbf{e}$ at distance e from a codeword $\mathbf{c}$ in the form $\mathbf{y} + \mathbf{a}$. As A is an e-error-correcting code, the only possibility is $\mathbf{e} = (\mathbf{e} - \mathbf{c}) + \mathbf{c}$, with $\mathbf{e} - \mathbf{c} \in Y_e$, hence $g \geq e$, as claimed. □

REMARKS. 1. The proposition is also due to MacWilliams and Delsarte. In MacWilliams' version for linear codes, $|\{\, m \neq 0 \,|\, b_m \neq 0 \,\}|$ is the number of nonzero weights in the dual code (as is evident from our proof of Theorem 9.4.8(2)).

2. We could combine Lloyd's Theorem and the proposition, with the rephrased proposition saying that equality holds if and only if A is perfect, in which case

the appropriate polynomial has the appropriate roots. This might shorten the proof somewhat but also make it more mysterious.

3. Recall that the covering radius $g = cr(A)$ of the code $A \subseteq F^n$ is the smallest g such that $F^n = \bigcup_{\mathbf{a}\in A} S_g(\mathbf{a})$. That is, it is the smallest g with $V = S_g + A$. A more careful proof of the proposition gives

$$|\{\, m \neq 0 \,|\, b_m \neq 0 \,\}| \geq cr(A)\,.$$

Lloyd's Theorem and the Sphere Packing Condition are the main tools used in proving nonexistence results for perfect codes and other related codes. The Sphere Packing Condition has a natural derivation in the present language:

$$|S_e|\,|C| = \chi(S_e|\mathbf{0})\chi(C|\mathbf{0}) = \chi(S_e \oplus C|\mathbf{0}) = \chi(V|\mathbf{0}) = |V|\,,$$

for the perfect e-error-correcting code C in V.

The following simple example of a nonexistence result for perfect codes is a good model for more general results of this type.

(9.4.11) THEOREM. *If C is a binary perfect 2-error-correcting code of length $n \geq 2$, then either $n = 2$ and $C = \mathbb{F}_2^2$ or $n = 5$ and $C = \{\, \mathbf{x}, \mathbf{y} \,|\, \mathbf{x} + \mathbf{y} = \mathbf{1} \,\}$.*

PROOF. We do this in three steps:

> Step 1. *Sphere Packing Condition:* There is a positive integer r with $2 + n + n^2 = 2^{r+1}$. If $n \leq 6$, then we have one of the examples of the theorem. Therefore we can assume that $n \geq 7$, and in particular $8n < 2^{r+1}$.

By the Sphere Packing Condition we have

$$1 + n + \binom{n}{2} = 2^r\,,$$

where r is the redundancy of the code. This simplifies to $2 + n + n^2 = 2^{r+1}$. We record the first few values:

n	2	3	4	5	6
$2 + n + n^2$	8	14	22	32	44

Only $n = 2, 5$ can occur, and in these cases we easily find the examples described. Therefore we may assume from now on that $n \geq 7$, in which case

$$2 + n(7 + 1) \leq 2 + n(n + 1) = 2^{r+1}\,.$$

> Step 2. *Lloyd's Theorem:* $n \equiv 3 \pmod 4$.

$$\begin{aligned}\Psi_2(x) &= K_1(x;n,2) + K_1(x;n,2) + K_2(x;n,2)\\ &= 1 + (n-2x) +\\ &\quad + \left((-1)^0\binom{x}{0}\binom{n-x}{2} + (-1)^1\binom{x}{1}\binom{n-x}{1} + (-1)^2\binom{x}{2}\binom{n-x}{0}\right)\\ &= \frac{1}{2}(4x^2 - 4(n+1)x + (2+n+n^2))\end{aligned}$$

By *Step 1.* $2+n+n^2 = 2^{r+1}$; so if we substitute y for $2x$, then Lloyd's Theorem 9.4.9 says that the quadratic polynomial

$$y^2 - 2(n+1)y + 2^{r+1}$$

has two even integer roots in the range 2 through $2n$. Indeed

$$y^2 - 2(n+1)y + 2^{r+1} = (y - 2^a)(y - 2^b)\,,$$

for positive integers a, b with $2^a + 2^b = 2(n+1)$.

We next claim that 2^a and 2^b can not be 2 or 4. If $y = 2$ is a root, then

$$4 - 2(n+1)2 + 2^{r+1} = 0 \;\text{ hence }\; 2^{r+1} = 4n < 8n\,.$$

Similarly if $y = 4$ is a root, then

$$16 - 2(n+1)4 + 2^{r+1} = 0 \;\text{ hence }\; 2^{r+1} = 8n - 8 < 8n\,;$$

and in both cases we contradict *Step 1.*

Therefore $a, b \geq 3$, and

$$n + 1 = 2^{a-1} + 2^{b-1} \equiv 0 \pmod 4\,,$$

as desired.

Step 3. *Conclusion.*

Let $n = 4m + 3$ and substitute into the Sphere Packing Condition:

$$\begin{aligned}2^{r+1} &= 2 + (4m+3) + (4m+3)^2\\ &= 2 + 4m + 3 + 16m^2 + 24m + 9\\ &= 14 + 28m + 16m^2\end{aligned}$$

The lefthand side is congruent to 0 modulo 4, while the righthand sice is congruent to 2 modulo 4. This contradiction completes the proof. □

REMARK. For $n = 2$, we find

$$y^2 - 2(n+1)y + 2^{r+1} = y^2 - 6y + 8 = (y-2)(y-4)\,;$$

and, for $n = 5$, we find

$$y^2 - 2(n+1)y + 2^{r+1} = y^2 - 12y + 32 = (y-4)(y-8)\,.$$

9.5 Generalizations of MacWilliams' Theorem

In Sections 9.2 and 9.3, for ease of exposition we only presented proofs of MacWilliams' Theorem 9.1.1 in the case of binary linear codes. On the other hand, in Section 9.4, once we had introduced the appropriate machinery, we were easily able to prove MacWilliams' Theorem for a class of codes larger than that of all linear codes over finite fields. It seems to have been Gleason (1971) who first fully appreciated the strength and generality of MacWilliams' proof using characters.

Initially in this section F is a ring that satisfies Property (ND) of Lemma 9.4.3, but we mainly concentrate on the case $F = \mathbb{Z}_4$ as it is of independent interest. We only give examples of two of the many generalizations that MacWilliams' Theorem admits.

Fourier transform

Let $V = F^n$, and let $f: V \longrightarrow R$ be a map to any vector space over $\mathbb{C}$. Then the (discrete) *Fourier transform* (or Hadamard transform) of f is $\hat{f}: V \longrightarrow R$ given by

$$\hat{f}(\mathbf{v}) = \sum_{\mathbf{u} \in V} f(\mathbf{u}) \chi(\mathbf{u}|\mathbf{v}) \,.$$

(9.5.1) Proposition. (Poisson summation formula.)
If A is an F-linear code then

$$\sum_{\mathbf{u} \in A^\perp} f(\mathbf{u}) = |A|^{-1} \sum_{\mathbf{v} \in A} \hat{f}(\mathbf{v}) \,.$$

Proof. We use Lemma 9.4.4(2) to calculate

$$\begin{aligned} |A|^{-1} \sum_{\mathbf{v} \in A} \hat{f}(\mathbf{v}) &= |A|^{-1} \sum_{\mathbf{v} \in A} \left(\sum_{\mathbf{u} \in V} f(\mathbf{u}) \chi(\mathbf{u}|\mathbf{v}) \right) \\ &= |A|^{-1} \sum_{\mathbf{u} \in V} f(\mathbf{u}) \left(\sum_{\mathbf{v} \in A} \chi(\mathbf{u}|\mathbf{v}) \right) \\ &= |A|^{-1} \sum_{\mathbf{u} \in A^\perp} f(\mathbf{u}) \, |A| \\ &= \sum_{\mathbf{u} \in A^\perp} f(\mathbf{u}) \,, \end{aligned}$$

as desired. □

This calculation was embedded in our proof of Theorem 9.4.8 for the particular choice of map $f(\mathbf{u}) = z^{w_H(\mathbf{u})} \in \mathbb{C}[z]$. Using the proposition and Proposition 9.4.6 in that case, we find

$$W_{A^\perp}(z) = \sum_{\mathbf{u} \in A^\perp} f(\mathbf{u}) = |A|^{-1} \sum_{\mathbf{v} \in A} \hat{f}(\mathbf{v})$$

$$\begin{aligned} &= |A|^{-1} \sum_{\mathbf{v}\in A} \left(\sum_{\mathbf{u}\in V} f(\mathbf{u})\chi(\mathbf{u}|\mathbf{v}) \right) \\ &= |A|^{-1} \sum_{\mathbf{v}\in A} (1+(s-1)z)^{n-w_H(\mathbf{v})}(1-z)^{w_H(\mathbf{v})} \\ &= |A|^{-1} \sum_{i=0}^{n} a_i (1+(s-1)z)^{n-i}(1-z)^i \,. \end{aligned}$$

This is MacWilliams' Theorem. (This can not really be described as a fourth proof but rather a rewording of the third proof.)

Remember that the homogeneous weight enumerator of A is

$$W_A(x,y) = \sum_{\mathbf{a}\in A} x^{n-w_H(\mathbf{a})} y^{w_H(\mathbf{a})} \,.$$

Thus, for a given $\mathbf{a} = (a_1, \ldots, a_j, \ldots, a_n)$, we have in $x^{n-w_H(\mathbf{a})}y^{w_H(\mathbf{a})}$ factors x, one for each $a_j = 0$, and factors y, one for each $a_j \neq 0$. Different nonzero coefficients make the same contribution. If instead we wish to note the contribution of each member of F, we look at the (homogeneous) *complete weight enumerator* $C_A(x_1, \ldots, x_s)$ for A, a polynomial in $s = |F|$ commuting variables, one to count occurrences of each member of the alphabet. (For the binary alphabet, the complete weight enumerator is just the usual homogeneous weight enumerator.)

complete weight enumerator

At this point we specialize to $F = \mathbb{Z}_4$. For the word $\mathbf{v} = (v_1, \ldots, v_j, \ldots, v_n)$ of V, we write $w_0(\mathbf{v})$ for the number of 0's among the v_j, $w_1(\mathbf{v})$ for the number of 1's, $w_2(\mathbf{v})$ for the number of 2's, and $w_3(\mathbf{v})$ for the number of 3's among the v_j. Thus $w_0(\mathbf{v}) + w_1(\mathbf{v}) + w_2(\mathbf{v}) + w_3(\mathbf{v}) = n$. The complete weight enumerator for A is then

$$C_A(x,y,z,w) = \sum_{\mathbf{a}\in A} x^{w_0(\mathbf{v})}\, y^{w_1(\mathbf{v})}\, z^{w_2(\mathbf{v})}\, w^{w_3(\mathbf{v})} \,.$$

If we want a version of MacWilliams' Theorem for complete weight enumerators, we do not have to retrace our steps in the previous sections. We just apply Poisson summation and the Fourier transform to a different base function f.

For $F = \mathbb{Z}_4$, we set $f(\mathbf{u}) = x^{w_0(\mathbf{u})}\, y^{w_1(\mathbf{u})}\, z^{w_2(\mathbf{u})}\, w^{w_3(\mathbf{u})}$. Then, as before, we have

$$C_{A^\perp}(x,y,z,w) = \sum_{\mathbf{u}\in A^\perp} f(\mathbf{u}) \;=\; |A|^{-1} \sum_{\mathbf{v}\in A} \hat{f}(\mathbf{v}) \,.$$

To compute the Fourier transform, we follow Proposition 9.4.6:

$$\begin{aligned} \hat{f}(\mathbf{v}) &= \sum_{\mathbf{u}\in V} f(\mathbf{u})\chi(\mathbf{u}|\mathbf{v}) \\ &= \sum_{\mathbf{u}\in V} x^{w_0(\mathbf{u})}\, y^{w_1(\mathbf{u})}\, z^{w_2(\mathbf{u})}\, w^{w_3(\mathbf{u})} \chi(\mathbf{u}|\mathbf{v}) \end{aligned}$$

$$\begin{aligned}
&= \sum_{\mathbf{u}\in V}\left(\prod_{j=1}^{n} x^{w_0(u_j)}\, y^{w_1(u_j)}\, z^{w_2(u_j)}\, w^{w_3(u_j)}\chi(u_j|v_j)\right)\\
&= \prod_{j=1}^{n}\left(\sum_{u\in F} x^{w_0(u)}\, y^{w_1(u)}\, z^{w_2(u)}\, w^{w_3(u)}\chi(u|v_j)\right)\\
&= \prod_{j=1}^{n}(x\,\chi(0|v_j)+y\,\chi(1|v_j)+z\,\chi(2|v_j)+w\,\chi(3|v_j))\,.
\end{aligned}$$

The value of each factor will depend upon that of v_j:

v_j	$x\,\chi(0\|v_j)+y\,\chi(1\|v_j)+z\,\chi(2\|v_j)+w\,\chi(3\|v_j)$
0	$x+y+z+w$
1	$x+iy-z-iw$
2	$x-y+z-w$
3	$x-iy-z+iw$

Hence finally

$$\begin{aligned}
\hat{f}(\mathbf{v}) &= (x+y+z+w)^{w_0(\mathbf{v})}\,(x+iy-z-iw)^{w_1(\mathbf{v})}\\
&\qquad (x-y+z-w)^{w_2(\mathbf{v})}\,(x-iy-z+iw)^{w_3(\mathbf{v})}\,.
\end{aligned}$$

When inserted into the summation formula, this gives MacWilliams' Theorem for complete weight enumerators over $\mathbb{Z}_4$:

(9.5.2) THEOREM. *If A is a $\mathbb{Z}_4$-linear code, then*

$$\begin{aligned}
C_{A^\perp}(x,y,z,w) &= |A|^{-1}\, C_A(x+y+z+w, x+iy-z-iw,\\
&\qquad\qquad x-y+z-w, x-iy-z+iw)\,. \qquad \square
\end{aligned}$$

Although this is quite complicated, it is also relatively powerful. For instance we regain the usual homogeneous $\mathbb{Z}_4$-version of MacWilliams' Theorem by specializing to $y=z=w$.

$$\begin{aligned}
W_{A^\perp}(x,y) &= C_{A^\perp}(x,y,y,y)\\
&= |A|^{-1}\, C_A(x+y+y+y, x+iy-y-iy,\\
&\qquad\qquad x-y+y-y, x-iy-y+iy)\\
&= |A|^{-1}\, C_A(x+3y, x-y, x-y, x-y)\\
&= |A|^{-1}\, W_A(x+3y, x-y)\,.
\end{aligned}$$

In certain situations there are metrics on F^n that are more appropriate than the Hamming metric. For instance, when reading time from a clock with hands but no numbers, it will be easier to mistake 3 o'clock for 2 or 4 than for 8 o'clock. The *Lee metric* on $\mathbb{Z}_s$ sets the Lee distance

Lee metric

$$d_L(i,j) = \min(|i-j|, s-|i-j|)\,,$$

for $i,j \in \mathbb{Z}_s$, and the Lee weight

$$w_L(i) = d_L(i,0)\,.$$

Thus, on $\mathbb{Z}_4$, we have

$$w_L(0) = 0,\ w_L(1) = w_L(3) = 1, \quad \text{and} \quad w_L(2) = 2\,.$$

This is the first new case, since the Lee metric on $\mathbb{Z}_2$ and $\mathbb{Z}_3$ is the same as the Hamming metric.

As before, two words $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{w} = (w_1, \ldots, w_n)$ from $\mathbb{Z}_s^n$ have Lee distance given by

$$d_L(\mathbf{v}, \mathbf{w}) = \sum_{j=1}^{n} d_L(v_j, w_j)$$

and Lee weight $w_L(\mathbf{v}) = d_L(\mathbf{v}, \mathbf{0})$.

The *Lee weight enumerator* of $A \subseteq \mathbb{Z}_s^n$ is then

Lee weight enumerator

$$L_A(z) = \sum_{\mathbf{v}\in A} z^{w_L(\mathbf{v})}\,.$$

As the largest weight of a word in $\mathbb{Z}_s^n$ is $t\,n$, where $t = \lfloor s/2 \rfloor$, the *homogeneous Lee weight enumerator* is

homogeneous Lee weight enumerator

$$L_A(x,y) = \sum_{\mathbf{v}\in A} x^{tn - w_L(\mathbf{v})} y^{w_L(\mathbf{v})}\,.$$

When $F = \mathbb{Z}_4$, the homogeneous Lee weight enumerator is

$$L_A(x,y) = \sum_{\mathbf{v}\in A} x^{2n - w_L(\mathbf{v})} y^{w_L(\mathbf{v})}\,.$$

For a particular word $\mathbf{v} \in \mathbb{Z}_4^n$ we see that

$$x^{2n-w_L(\mathbf{v})} y^{w_L(\mathbf{v})} = (x^2)^{w_0(\mathbf{v})} (xy)^{w_1(\mathbf{v})} (y^2)^{w_2(\mathbf{v})} (xy)^{w_3(\mathbf{v})}\,,$$

and therefore

$$L_A(x,y) = C_A\big(x^2, xy, y^2, xy\big)\ .$$

(9.5.3) THEOREM. *If A is a $\mathbb{Z}_4$-linear code, then*

$$L_{A^\perp}(x,y) = |A|^{-1} L_A(x+y, x-y)\ .$$

PROOF. We use Theorem 9.5.2.

$$\begin{aligned}
L_{A^\perp}(x,y) &= C_{A^\perp}\big(x^2, xy, y^2, xy\big) \\
&= |A|^{-1}\, C_A\big(x^2 + xy + y^2 + xy, x^2 + ixy - y^2 - ixy, \\
&\qquad\qquad x^2 - xy + y^2 - xy, x^2 - ixy - y^2 + ixy\big) \\
&= |A|^{-1}\, C_A\big((x+y)^2, x^2 - y^2, (x-y)^2, x^2 - y^2\big) \\
&= |A|^{-1}\, L_A(x+y, x-y)\ . \qquad \square
\end{aligned}$$

It is notable (and significant) that the transformation

$$x \longrightarrow x + y \qquad y \longrightarrow x - y\,,$$

which takes the homogeneous Lee weight enumerator of the $\mathbb{Z}_4$-linear code A to $|A|$ times that of its dual, is the same transformation that takes the homogeneous weight enumerator of the binary linear code A to $|A|$ times that of its dual. (See Theorem 9.1.1(2).)

Appendix A

Some Algebra

This appendix is broken into three sections.

The first section discusses doing basic arithmetic and vector algebra over arbitrary fields of coefficients, rather than restricting to the usual rational, real, or complex fields. Anyone who has had contact with some abstract algebra at college level should be comfortable with this material. Those already familiar with it should be aware that the section contains some of the definitions that we shall be using, so it can not be skipped entirely.

The second section deals with the fundamentals of polynomial algebra with coefficients from an arbitrary field. This material is more advanced but most should again look familiar, particularly to students who have had an undergraduate level abstract algebra sequence.

The final section covers more specialized topics that may not seem familiar.

A.1. Basic Algebra

A.1.1. Fields

In doing coding theory it is advantageous for our alphabet to have a certain amount of mathematical structure. We are familiar at the bit level with boolean addition (EXCLUSIVE OR) and multiplication (AND) within the set $\{0,1\}$:

$+$	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

We wish to give other alphabets, particularly finite ones, a workable arithmetic. The objects we study (and of which the set $\{0,1\}$ together with the above operations is an example) are called fields. A field is basically a set that possesses an arithmetic having (most of) the properties that we expect — the ability to add, multiply, subtract, and divide subject to the usual laws of commutativity, associativity, and distributivity. The typical examples are the field of rational numbers (usually denoted $\mathbb{Q}$), the field of real numbers $\mathbb{R}$, and the field of complex numbers $\mathbb{C}$; however as just mentioned not all examples are so familiar. The integers do *not* constitute a field because in general it is not possible to divide one integer by another and have the result still be an integer.

field

A *field* is, by definition, a set F, say, equipped with two operations, $+$ (addition) and $\cdot$ (multiplication), which satisfy the following seven usual arithmetic axioms:

(1) (Closure) For each a and b in F, the sum $a+b$ and the product $a \cdot b$ are well-defined members of F.

(2) (Commutativity) For all a and b in F, $a+b = b+a$ and $a \cdot b = b \cdot a$.

(3) (Associativity) For all a, b, and c in F, $(a+b)+c = a+(b+c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(4) (Distributivity) For all a, b, and c in F, $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(a+b) \cdot c = a \cdot c + b \cdot c$.

(5) (Existence of identity elements) There are distinct elements 0 and 1 of F such that, for all a in F, $a+0 = 0+a = a$ and $a \cdot 1 = 1 \cdot a = a$.

(6) (Existence of additive inverses) For each a of F there is an element $-a$ of F such that $a + (-a) = (-a) + a = 0$.

(7) (Existence of multiplicative inverses) For each a of F that does not equal 0, there is an element a^{-1} of F such that $a \cdot (a^{-1}) = (a^{-1}) \cdot a = 1$.

It should be emphasized that these common arithmetic assumptions are the only ones we make. In particular we make no flat assumptions about operations

called "subtraction" or "division". These operations are best thought of as the "undoing" respectively of addition and multiplication and, when desired, can be defined using the known operations and their properties. Thus subtraction can be defined by $a - b = a + (-b)$ using (6), and division defined by $a/b = a \cdot (b^{-1})$ using (7) (provided b is not 0).

Other familiar arithmetic properties that are not assumed as axioms either must be proven from the assumptions or may be false in certain fields. For instance, it is not assumed but can be proven that always in a field $(-1)\cdot a = -a$. (Try it!) A related, familiar result which can be proven for all fields F is that, given a and b in F, there is always a unique solution x in F to the equation $a+x = b$. On the other hand the properties of positive and/or negative numbers familiar from working in the rational field $\mathbb{Q}$ and the real field $\mathbb{R}$ do not have a place in the general theory of fields. Indeed there is no concept at all of "negative" or "positive" number for the complex field $\mathbb{C}$ or the field $\{0, 1\}$ discussed above.

The only thing keeping the integers $\mathbb{Z}$ from being a field is the axiom (7) concerning multiplicative inverses. Axioms (1)-(6) are valid for $\mathbb{Z}$, but (7) fails miserably; indeed 1 and -1 are the *only* integers that possess multiplicative inverses that are also integers. The integers do satisfy two axioms weaker than (7) but still useful.

(7′) (Cancellation) If a is not 0 and $a \cdot b = a \cdot c$, then $b = c$.

(7″) (No Zero Divisors) If $a \cdot b = 0$, then $a = 0$ or $b = 0$.

Axiom (7′) is a direct consequence of (7), because multiplying both sides of $a \cdot b = a \cdot c$ by a^{-1} leaves $b = c$. However (7) is not a consequence of (7′) as (7′) is true in $\mathbb{Z}$ while (7) is not. Similarly axiom (7″) is a consequence of (7). If one of a or b is not zero, then multiplying the lefthand side of $a \cdot b = 0$ by its inverse reveals the other as equal to 0. Again (7″) is true in $\mathbb{Z}$ while (7) is not, so that (7) is not a consequence of (7″).

In fact axioms (7′) and (7″) are equivalent in the following sense: if the set R has operations $+$ and $\cdot$ that satisfy (1) through (6), then either both axioms (7′) and (7″) hold or neither does. To see that (7′) implies (7″), apply (7′) to $a \cdot b = a \cdot 0$. On the other hand, to see that (7″) implies (7′), apply (7″) to $a \cdot (b - c) = 0$.

We are interested mainly in *finite fields*, those fields with a finite number of elements of which $\{0, 1\}$ is our only example so far. The most familiar way of giving a reasonable arithmetic to a finite set is to do modular arithmetic in the integers. For a fixed positive integer n, called the *modulus* we give the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ an arithmetic by first performing the usual integer addition or multiplication and then reducing the result modulo n back into the set $\mathbb{Z}_n$ by subtracting off multiples of n. This is "clock arithmetic" when $n = 12$ (or, these days, when $n = 24$).

finite fields

modulus

The question arises as to whether $\mathbb{Z}_n$ is a field. The field $\{0, 1\}$ already mentioned several times is nothing other than the integers mod 2, $\mathbb{Z}_2$. It is not difficult to check that $\mathbb{Z}_n$ with modular arithmetic satisfies axioms (1) through

(6). On the other hand, the answer as to whether $\mathbb{Z}_n$ satisfies (7) or the weaker (7′) and (7″) depends upon the particular value of the modulus n. For instance, all are true when $n = 2$. For $n = 6$ we have $2 \cdot 3 = 0 \pmod 6$ (whence $2{\cdot}3 = 2{\cdot}0 \pmod 6$); yet neither 2 nor 3 equals 0 in the integers mod 6. Therefore each of (7), (7′), and (7″) is false in $\mathbb{Z}_6$.

Although the arithmetics of $\mathbb{Z}$ and $\mathbb{Z}_6$ do not make them into fields, the structures clearly are of interest. A set F equipped with an addition and multiplication that satisfy (1) through (6) we shall call a *commutative ring*. ("Commutative" because the multiplication satisfies the commutative law.) A *ring* satisfies each of (1) through (6) with the possible exception of the commutativity of multiplication. If the commutative ring F additionally satisfies the equivalent axioms (7′) and (7″), then it is called an *integral domain* (in honor of the integers!). Clearly all fields and all integral domains are commutative rings. As (7) implies (7′) and (7″), every field is also an integral domain while the integers provide the prime example of an integral domain that is not a field. $\mathbb{Z}_6$ is an example of a commutative ring that is not an integral domain and so certainly not a field.

commutative ring
ring
integral domain

An element of a ring that has an inverse, as in (7), is called a *unit*; so fields are exactly those commutative rings in which every nonzero element is a unit.

unit

(A.1.1) LEMMA. *Let n be an integer larger than 1. The following are equivalent:*

(1) *n is a prime;*
(2) *$\mathbb{Z}_n$ is an integral domain;*
(3) *$\mathbb{Z}_n$ is a field.*

PROOF. *(1) implies (2)*: Assume n is a prime, and that $a \cdot b = 0$ in $\mathbb{Z}_n$. Then the integer ab is a multiple of n. As n is prime, it divides either a or b; hence either a or b is 0 in $\mathbb{Z}_n$. This verifies axiom (7″).

(2) implies (1): As with our example of $\mathbb{Z}_6$, if n is not prime, then each factorization $ab = n$ in $\mathbb{Z}$ with $1 < a, b < n$ gives rise to an equality $a \cdot b = 0$ in $\mathbb{Z}_n$ with neither a nor b equal to 0. Thus if n is not a prime, then $\mathbb{Z}_n$ does not satisfy (7″) and so is not an integral domain.

(3) implies (2) as axiom (7) implies axioms (7′) and (7″).

(2) implies (3): Let $\mathbb{Z}_n^\sharp = \{1, \ldots, n-1\}$, the set of nonzero elements of $\mathbb{Z}_n$. Choose $a \in \mathbb{Z}_n^\sharp$. As (by assumption) $\mathbb{Z}_n$ is an integral domain, for distinct elements $z_1, z_2 \in \mathbb{Z}_n^\sharp$, the products $a \cdot z_1$ and $a \cdot z_2$ are also distinct by (7′). Therefore the set $a\mathbb{Z}_n^\sharp = \{a \cdot z | z \in \mathbb{Z}_n^\sharp\}$ contains $n-1$ distinct members of $\mathbb{Z}_n$. Indeed $0 \notin a\mathbb{Z}_n^\sharp$ by (7″), so $a\mathbb{Z}_n^\sharp$ is a subset of $\mathbb{Z}_n^\sharp$. Thus $a\mathbb{Z}_n^\sharp$ is a subset of $\mathbb{Z}_n^\sharp$ containing the same number of elements as $\mathbb{Z}_n^\sharp$. We conclude that $\mathbb{Z}_n^\sharp = a\mathbb{Z}_n^\sharp$. In particular, $1 \in \mathbb{Z}_n^\sharp = a\mathbb{Z}_n^\sharp$; and there is a z in $\mathbb{Z}_n^\sharp$ with $a \cdot z = 1$. Therefore all the nonzero members of $\mathbb{Z}_n$ have multiplicative inverses in $\mathbb{Z}_n$, and $\mathbb{Z}_n$ is a field. □

(A.1.2) PROBLEM. *Extend the argument of Lemma A.1.1 that (2) implies (3) to prove the more general result that every finite integral domain is in fact a field. (The integers of course provide an example of an infinite integral domain that is not a field.)*

If F is a field and K is a subset of F, K is said to be a *subfield* of F provided that the set K equipped with the addition and multiplication of F is a field in its own right. If this is the case, then we write $K \leq F$ or $F \geq K$. The addition and multiplication of K will be commutative, associative, and distributive as they already are in F; so the crucial requirements are that K be closed under addition and multiplication and contain the additive and multiplicative inverses of all its elements. As examples, the rational field $\mathbb{Q}$ is a subfield of the real field $\mathbb{R}$, which in turn is a subfield of the complex field $\mathbb{C}$.

subfield

If K is a subfield of F, then we call F an *extension field* of K. Thus $\mathbb{C}$ is an extension field of $\mathbb{R}$, and both $\mathbb{C}$ and $\mathbb{R}$ are extension fields of $\mathbb{Q}$. As we shall mainly be concerned with finite fields, important examples of subfields for us are provided by the next result.

extension field

(A.1.3) LEMMA. *Let F be a finite field, and consider the subset K of F composed of all elements of F that can be written as a sum of 1's:*

$$K = \{1,\ 1+1,\ 1+1+1,\ 1+1+1+1,\ \dots\}.$$

Then K is a subfield $\mathbb{Z}_p$ of F, for some prime p.

PROOF. Notice that by definition K is closed under addition, while an easy application of the distributive law in F shows that K is closed under multiplication.

As F is finite, so is K. Therefore there are distinct positive integers m and n (m larger than n) with the sum of m 1's equal to the sum of n 1's. (Indeed, there are many such pairs m, n.) Equivalently the sum of $m-n$ 1's equals 0 in F and K, $m-n$ a positive integer. Let p be the smallest positive integer such that 0 is a sum of p 1's in F and K. We conclude that K is composed precisely of the p distinct elements

$$1,\ 1+1,\ \dots,\ \sum_{i=1}^{p} 1 = \overbrace{1+\cdots+1}^{p \text{ times}} = 0.$$

The set K is therefore a copy of $\mathbb{Z}_p$. As K is contained in the field F, no two nonzero members of K have product 0; so by Lemma A.1.1 p is a prime, completing the result. □

The prime p of Lemma A.1.3 is called the *characteristic* of the field F, and K is (for obvious reasons) called the *prime subfield* of F.

characteristic
prime subfield

A.1.2. Vector spaces

The passage from the real line to real, Euclidean, three-dimensional space is the most familiar case of the passage from a field to a vector space over a field. If F is a field and n is any positive integer, we may use the arithmetic structure of F to give the set F^n of n-tuples from F,

$$F^n = \{(a_1, a_2, \ldots, a_n) \,|\, a_i \in F\},$$

additive and multiplicative structures as well. We define "vector addition" of members of F^n via

$$(a_1, a_2, \ldots, a_n) \oplus (b_1, b_2, \ldots, b_n) = (c_1, c_2, \ldots, c_n)$$

where $c_i = a_i + b_i$ (addition in F), for each $i = 1, \ldots, n$. We define "scalar multiplication" of members of F^n by members of F via

$$\alpha \star (a_1, a_2, \ldots, a_n) = (\alpha \cdot a_1, \alpha \cdot a_2, \ldots, \alpha \cdot a_n)$$

where $\alpha \cdot a_i$ is the usual multiplication in the field F. These two operations make F^n into a vector space over F.

vector space

Given a field F, a *vector space* V *over* F is, by definition, a set V (whose members are called the *vectors* of V) equipped with two operations $\oplus$ (vector addition) and $\star$ (scalar multiplication), satisfying the following:

(1) (Closure) For each $\mathbf{v}$ and $\mathbf{w}$ in V, $\mathbf{v} \oplus \mathbf{w}$ is a well-defined member of V. For each α in F and $\mathbf{v}$ in V, $\alpha \star \mathbf{v}$ is a well-defined member of V.

(2) (Commutativity) For each $\mathbf{v}$ and $\mathbf{w}$ in V, $\mathbf{v} \oplus \mathbf{w} = \mathbf{w} \oplus \mathbf{v}$.

(3) (Associativity) For each $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in V, $(\mathbf{u} \oplus \mathbf{v}) \oplus \mathbf{w} = \mathbf{u} \oplus (\mathbf{v} \oplus \mathbf{w}$. For each α, β in F and $\mathbf{v}$ in V, $(\alpha \cdot \beta) \star \mathbf{v} = \alpha \star (\beta \star \mathbf{v})$.

(4) (Distributivity) For each α, β in F and $\mathbf{v}, \mathbf{w}$ in V, $(\alpha + \beta) \star \mathbf{v} = (\alpha \star \mathbf{v}) \oplus (\beta \star \mathbf{v})$ and $\alpha \star (\mathbf{v} \oplus \mathbf{w}) = (\alpha \star \mathbf{v}) \oplus (\alpha \star \mathbf{w})$.

(5) (Existence of vector identity) There is a vector $\mathbf{0}$ of V such that, for each $\mathbf{v}$ of V, $\mathbf{v} \oplus \mathbf{0} = \mathbf{0} \oplus \mathbf{v} = \mathbf{v}$.

(6) (Existence of vector inverses) For each $\mathbf{v}$ of V there is a vector $-\mathbf{v}$ of V such that $\mathbf{v} \oplus (-\mathbf{v}) = (-\mathbf{v}) \oplus \mathbf{v} = \mathbf{0}$.

(7) (Scalar identity properties) For each $\mathbf{v}$ of V, $1 \star \mathbf{v} = \mathbf{v}$ and $0 \star \mathbf{v} = \mathbf{0}$.

F-space

For brevity, we sometimes say that V is an *F-vector space* or even an *F-space*. Note that scalar multiplication $\star$ is not multiplication of one vector by another but multiplication of a vector in V by a member of the field F. (F is usually called the *scalar field* of the vector space V, and its members are *scalars*.)

scalar field
scalars

The set F^n with the operations defined above is now easily seen to be a vector space over F. The similarity between the above axioms and those of Section A.1.1 explains the fact that F may be thought of as a vector space over itself. (After all, the distinction between F and F^1 is merely a pair of parentheses.) Many examples of vector spaces do not resemble the space of n-tuples at all. For instance, the set of all continuous and differentiable functions on the real line is a vector space over the real numbers.

Most of the vector spaces we shall study will naturally sit inside vector spaces F^n (because the spaces F^n are the natural universes for the codes we study). A subset W of the vector space V over F is a *subspace* of V if the operations of V give W the structure of a vector space over F in its own right. In this case we shall write $W \leq V$ or $V \geq W$. As most of the axioms (2)-(7) will have already been checked within V, the main force of this definition is in the assumption that W is closed as in (1). In fact, the subset W of V will be a subspace of V if and only if, for all α in F and all $\mathbf{v}, \mathbf{w}$ in W, $\alpha \star \mathbf{v}$ is in W and $\mathbf{v} \oplus \mathbf{w}$ is in W. Thus V itself and $\{\mathbf{0}\}$ are rather trivial subspaces of V. More typical is the subspace of F^n composed of all vectors $(a_1, a_2, \ldots, a_n)$ with $a_1 + a_2 + \cdots + a_n = 0$. subspace

(A.1.4) PROBLEM. *Prove that the nonempty subset W of the F-vector space V is a subspace if and only if $\alpha\mathbf{v} + \mathbf{w} \in W$, for all $\mathbf{v}, \mathbf{w} \in W$ and $\alpha \in F$.*

If W is a subspace of V, then a *cosets* of W in V is a translate of W by some fixed vector. If we translate each vector of W by the vector $\mathbf{v}$, we get the coset $\mathbf{x} + W = \{\mathbf{x} + \mathbf{w} \mid \mathbf{w} \in W\}$. You should convince yourself that if $\mathbf{y} \in \mathbf{x} + W$, then $\mathbf{y} + W = \mathbf{x} + W$; so two cosets are either disjoint or equal. As an example, a typical subspace of dimension 2 in 3-dimensional Euclidean space is a plane through the origin, while a typical coset is a translate of such a subspace and so is a plane that need not be through the origin. cosets

One way of constructing a subspace of the F-vector space V is by taking the *span* $\langle S \rangle$ of a nonempty subset S of V. This is, by definition, the smallest subspace of V that contains S; however this may not be the best way of thinking of $\langle S \rangle$. We usually view $\langle S \rangle$ instead as the subspace composed of all linear combinations of members of S: span

$$\langle S \rangle = \{\sum_{\mathbf{v} \in S} \alpha_{\mathbf{v}} \mathbf{v} \mid \alpha_{\mathbf{v}} \in F\}.$$

You should convince yourself that these two definitions of $\langle S \rangle$ are equivalent. If $V = \langle S \rangle$, then S is a called a *spanning set* in V. spanning set

A *basis* of the vector space V is a minimal spanning set for V, a set that spans V but no proper subset of it spans V. basis

(A.1.5) THEOREM. *If the vector space V has a finite basis $\mathcal{B}$, then every basis of V contains the same number of vectors as $\mathcal{B}$.*

This theorem will be proven in the following subsection. (The theorem is in fact true without the assumption that $\mathcal{B}$ is finite.) The common size for the bases of V is the *dimension* of V. dimension

linearly dependent

A set Δ of vectors from V is called *linearly dependent* if there is a set of coefficients $\alpha_{\mathbf{v}}$, for $\mathbf{v} \in \Delta$, such that the linear combination $\sum_{\mathbf{v}\in\Delta} \alpha_{\mathbf{v}}\mathbf{v}$ equals $\mathbf{0}$. The equation

$$\sum_{\mathbf{v}\in\Delta} \alpha_{\mathbf{v}}\mathbf{v} = \mathbf{0}$$

linear dependence

is then called a *linear dependence* of Δ.

linearly independent

A subset Δ is *linearly independent* if it is not linearly dependent. The maximal linearly independent subsets of V are precisely the bases of V. (Check!) In particular, every linearly independent subset of V belongs to a basis. (For infinite dimensional spaces, this is the best way to see that a basis exists.)

(A.1.6) PROBLEM.

(a) Let E be an extension field of F. Prove that E is a vector space over F with scalar multiplication induced by the field multiplication of E.

(b) Using (1), show that every finite field has a prime power number of elements.

$GF(q)$ $\mathbb{F}_q$

If q is a power of a prime, we often write $GF(q)$ or $\mathbb{F}_q$ for a field containing q elements.

REMARKS ON NOTATION

Notice that in vector spaces we have two concepts of "addition" ($+$ in F and $\oplus$ in V) and two of "multiplication" ($\cdot$ in F and $\star$ in V) and that for formal precision we must distinguish between them. (See, for instance, axioms (3) and (4) above.) Often to simplify notation we adopt the usual practice of denoting all forms of addition by $+$ and all forms of multiplication by juxtaposition; so for α, β in F and $\mathbf{v}, \mathbf{w}$ in V we usually write

$$\alpha\beta \text{ for } \alpha \cdot \beta \text{ ; } \mathbf{v} + \mathbf{w} \text{ for } \mathbf{v} \oplus \mathbf{w} \text{ ; and } \alpha\mathbf{v} \text{ for } \alpha \star \mathbf{v}.$$

In doing this we risk ambiguity. To counter this possibility we often adopt other conventions which may already have been noticed. For instance, we usually write vectors in boldface thus: $\mathbf{v}$.

A.1.3. Matrices

Just as we can examine vector spaces over arbitrary fields, so can we define matrices with entries from an arbitrary field. If K is a field, we denote by $K^{m,n}$ the collection of all $m \times n$ matrices with entries from K. Notice that the vector space K^n of row vectors of length n is equal to $K^{1,n}$. The vector space of column vectors of length m is $K^{m,1}$. The usual componentwise addition and subtraction is defined on $K^{m,n}$ and has all the expected properties. Together with scalar multiplication, these give $K^{m,n}$ the structure of a vector space over K of dimension mn.

Matrix multiplication is also defined by the familiar formula (*i.e.*, entries of the product being the dot product of rows of the first matrix with columns of the second). Matrix multiplication also has all the expected properties — associativity, distributivity over addition, block multiplication. Because the most usual matrix manipulations involve only addition, subtraction, and multiplication, the entries need not always be restricted to a field but might instead be from an integral domain (or even a ring).

You may notice that the set $K^{n,n}$ of square matrices together with the operations of matrix addition and multiplication satisfies all the axioms (1) through (6) with the exception of commutativity of multiplication. Thus $K^{n,n}$ is an example of a noncommutative ring.

If A is an $m \times n$ matrix with entries from the field K, then the *row space* of A, $\mathbf{RS}(A)$, is the subspace of K^n that is spanned by the rows of A. (We shall often look at codes that have been defined as the row space of certain matrices.) Similarly the *column space* of A, $\mathbf{CS}(A)$, is the subspace of $K^{m,1}$ spanned by the columns of A. The *null space* of A, $\mathbf{NS}(A)$, is the space of column vectors $\mathbf{x} \in K^{n,1}$ such that $A\mathbf{x} = \mathbf{0}$. (Notice that the null space can be thought of as the space of all linear dependencies on the set of columns.) The dimension of the row space of A is the *row rank* of A, and the dimension of the column space of A is the *column rank* of A. The dimension of $\mathbf{NS}(A)$ is the *nullity* of A.

row space
column space
null space
row rank
column rank
nullity

More complicated but familiar matrix processes can also be done over arbitrary fields. In particular, Gauss-Jordan elimination is still available. That is, by sequences of elementary row operations on a matrix it is possible to transform the matrix into reduced row echelon form. Several of the standard consequences of Gaussian elimination then become available. In particular we have:

(A.1.7) THEOREM. *Let A be an $m \times n$ matrix with entries from the field K.*

(1) *The row rank of A equals the column rank of A. (This common dimension being called the* rank *of A.)*

rank

(2) *The rank of A plus the nullity of A equals n, the number of columns of A.*

Before proving this theorem, we give a detailed discussion of echelon form and its properties. The *leading entry* of a row is its first nonzero entry, reading from left to right.

leading entry

The matrix A is said to be in *row echelon form* if it satisfies:

row echelon form

(1) the leading entry of each row is to the right of the leading entries of previous rows;

(2) all rows composed entirely of zeros are at the bottom of the matrix.

reduced row echelon form $\mathcal{RREF}$

The matrix A is said to be in *reduced row echelon form* $\mathcal{RREF}$, if it additionally satisfies:

(3) the leading entry of each row equals 1 and is the only nonzero entry in its column.

pivot entries
pivot columns

The various leading entries of the matrix $\mathcal{RREF}(A)$ are also sometimes called the *pivot entries* of $\mathcal{RREF}(A)$ and the columns containing them are the *pivot columns* of $\mathcal{RREF}(A)$ and A. The row rank of $\mathcal{RREF}(A)$ (indeed any matrix in row echelon form) is particularly easy to calculate; it is just the number of nonzero rows. It only takes a few seconds more to realize that this is also equal to the column rank of $\mathcal{RREF}(A)$. We will reduce the proof of Theorem A.1.7 to this special case, where we have just seen that the theorem (or at least its first part) is evident.

Elementary row operations have one of three forms:

(i) subtracting a multiple of one row from another;

(ii) interchanging two rows;

(iii) multiplying a row by a nonzero constant.

The usual elimination techniques then give:

(A.1.8) THEOREM. *Every matrix A with entries from a field can be transformed by a sequence of elementary row operations into a matrix $\mathcal{RREF}(A)$ that is in reduced row echelon form.* □

The verification is routine, but it is important that the matrix entries are from a field. For more general rings the result may not be true. (Imagine what could be done by integer row operations to a matrix with entries from $\mathbf{Z}$ whose first column contained only even integers.)

The notation suggests that $\mathcal{RREF}(A)$ is uniquely determined. This is indeed the case.

(A.1.9) PROBLEM. *Prove that the matrix $A \in K^{m,n}$, K a field, has a unique row reduced echelon form. (*HINT: *Prove that every vector of $\mathbf{RS}(A)$ has its leftmost nonzero entry in a pivot column, then either (i) try to write the rows of a second $\mathcal{RREF}$ as linear combinations of the rows of the first, or (ii) observe that the pivot columns are the leftmost columns that form a basis for $\mathbf{CS}(A)$.)*

As expected, each elementary row operation can be accomplished through left multiplication by an appropriate elementary matrix. Let $a\epsilon_{i,j}$ be the matrix that has a in its (i,j)-entry and 0's elsewhere (and write $\epsilon_{i,j}$ for $1\epsilon_{i,j}$), and let I be the identity matrix. Then left multiplication by

(i) $I + a\epsilon_{i,j}$ adds a times row j to row i;

(ii) $I - \epsilon_{i,i} - \epsilon_{j,j} + \epsilon_{i,j} + \epsilon_{j,i}$ interchanges rows i and j;

(iii) $I + (a-1)\epsilon_{i,i}$ multiplies row i by a.

The inverse of $I + a\epsilon_{i,j}$ is $I - a\epsilon_{i,j}$; $I - \epsilon_{i,i} - \epsilon_{j,j} + \epsilon_{i,j} + \epsilon_{j,i}$ is its own inverse; and $I + (a^{-1} - 1)\epsilon_{i,i}$ is the inverse of $I + (a-1)\epsilon_{i,i}$ for nonzero a. Therefore each elementary matrix is invertible. In particular we have $XA = \mathcal{RREF}(A)$, where the invertible matrix X is the product of those elementary matrices that correspond to the elementary row operations that take A to $\mathcal{RREF}(A)$.

(A.1.10) PROBLEM. *Let Y be an invertible $k \times k$ matrix with entries from the field K, and let A be the $k \times 2k$ matrix $\Big(Y \,|\, I\Big)$, the columns of Y followed by the columns of a $k \times k$ identity matrix. Prove that $\mathcal{RREF}(A) = \Big(I \,|\, Y^{-1}\Big)$.*

(A.1.11) PROBLEM. *Let Y be a $k \times k$ matrix with entries from the field K. Prove that the following are equivalent:*
(a) *Y is invertible;*
(b) $\mathbf{NS}(Y) = \mathbf{0}$*;*
(c) *Y has rank k;*
(d) $\mathcal{RREF}(Y) = I$.

(A.1.12) PROPOSITION. *Let A be an $m \times n$ matrix with entries from the field K.*
(1) *The column rank of $\mathcal{RREF}(A)$ equals the row rank of $\mathcal{RREF}(A)$.*
(2) $\mathbf{RS}(\mathcal{RREF}(A)) = \mathbf{RS}(A)$;
(3) $\mathbf{NS}(\mathcal{RREF}(A)) = \mathbf{NS}(A)$;
(4) $\dim(\mathbf{CS}(\mathcal{RREF}(A)) = \dim(\mathbf{CS}(A))$, *that is, $\mathcal{RREF}(A)$ has column rank equal to the column rank of A.*

PROOF. (1) Both numbers are equal to the number of pivot entries in $\mathcal{RREF}(A)$. Each of (2), (3), and (4) can be proven using the fact that there is an invertible matrix X with $XA = \mathcal{RREF}(A)$. For (4) it should be noted that (whether X is invertible or not) we have $\mathbf{CS}(XA) = X\mathbf{CS}(A) = \{\, X\mathbf{a} \,|\, \mathbf{a} \in \mathbf{CS}(A) \,\}$. □

(A.1.13) PROBLEM. *Prove completely parts (2), (3), and (4) of the proposition. Give an example that shows that $\mathbf{CS}(\mathcal{RREF}(A))$ and $\mathbf{CS}(A)$ need not be equal.*

If Σ is a set of vectors in F^n, then we can easily find a basis for $\langle\Sigma\rangle$ by forming the matrix A whose rows are the members of Σ and then passing to $\mathcal{RREF}(A)$ with its nonzero rows giving the desired basis. This observation is the basis for our proof of Theorem A.1.5: a vector space V with a finite basis $\mathcal{B}$ has all of its bases of size $|\mathcal{B}|$.

PROOF OF THEOREM A.1.5.
Choose $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ to be a basis for V of smallest size (necessarily finite). Let $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_d, \ldots\}$ be a second basis of V. Note that $|\mathcal{C}| \geq d = |\mathcal{B}|$

by choice. (If we encounter a second basis that is infinite, then for $\mathcal{C}$ we instead choose a finite subset of the basis that has at least $d+1$ members.)

For each i, write $\mathbf{c}_i$ as a linear combination of members of $\mathcal{B}$:

$$\mathbf{c}_i = \sum_{j=1}^{d} a_{i,j}\mathbf{b}_j\,;$$

and let the matrix A have (i,j)-entry $a_{i,j}$. As $\mathcal{C}$ is linearly independent, the row rank of A equals $|\mathcal{C}|$. However by Proposition A.1.12 the row rank of A equals the row rank of $\mathcal{RREF}(A)$ which is at most d, the number of columns of A. Therefore $|\mathcal{C}| \leq d$, completing the proof. □

For any matrix A, another advantage to having $R = \mathcal{RREF}(A)$ available is the ease with which its null space (and so that of A) can be calculated. Let the (i,j)-entry of R be $r_{i,j}$, and assume that the pivot entries are $r_{i,p(i)} = 1$, for $i = 1,\ldots,r$, (r being the row rank of A). Set $\mathcal{P} = \{\, p(i) \mid i = 1,\ldots,r\}$, the indices of the pivot columns of R.

For each nonpivot column $k \notin \mathcal{P}$ we construct a null vector $\mathbf{n}_k$ of R with a 1 in position k and 0 in all other nonpivot columns. The j-entry of $\mathbf{n}_k$ is given by:

$$\begin{array}{ll} (\mathbf{n}_k)_j = 1 & \text{if } j = k; \\ (\mathbf{n}_k)_j = 0 & \text{if } j \neq k \text{ and } j \notin \mathcal{P}; \\ (\mathbf{n}_k)_j = -r_{i,k} & \text{if } j = p(i) \in \mathcal{P}. \end{array}$$

This produces $n - r$ linearly independent vectors of $\mathbf{NS}(R)$. It is easy to see that $\mathbf{0}$ is the only null vector of R (and A) that is 0 in all nonpivot columns. Thus $\{\mathbf{n}_k \mid k \notin \mathcal{P}\}$ is a basis of $\mathbf{NS}(R) = \mathbf{NS}(A)$.

(A.1.14) PROBLEM. *Check that each $\mathbf{n}_k$ is indeed a null vector of R, and supply the remaining details of the proof that these vectors form a basis for $\mathbf{NS}(R)$.*

In particular we have just proven that the nullity of A is equal to the number of nonpivot columns in $\mathcal{RREF}(A)$. This together with Proposition A.1.12 allows us to prove Theorem A.1.7 easily.

PROOF OF THEOREM A.1.7.
For part (1), we have:

$$\begin{array}{rcl} \text{row rank of } A & = & \text{row rank of } \mathcal{RREF}(A) \text{ by A.1.12(2)} \\ & = & \text{column rank of } \mathcal{RREF}(A) \text{ by A.1.12(1)} \\ & = & \text{column rank of } A \text{ by A.1.12(4).} \end{array}$$

For part (2), if n is the number of columns in A (and $\mathcal{RREF}(A)$), then

$$\begin{array}{rcl} \text{rank of } A & = & \text{number of pivot columns in } \mathcal{RREF}(A) \text{ by A.1.12(1)} \\ & = & n \text{ minus number of nonpivot columns in } \mathcal{RREF}(A) \\ & = & n \text{ minus the nullity of } A \text{ by the above.} \end{array}$$

Thus both parts of the theorem are proved. □

We are familiar with the fact that division by matrices is a much trickier process than the other three arithmetic operations. In particular some concept of determinant is usually needed to talk about matrix inverses. Again the usual theory of determinants carries over to square matrices over arbitrary fields (and even rings). The standard formula gives a multiplicative function from $K^{n,n}$ into K. We shall have little need for this and leave the most elementary case as an exercise.

(A.1.15) PROBLEM. *For a* 2×2 *matrix* $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ *with entries from a commutative ring* R*, we define the* determinant $\det(A) = ad - bc$. determinant

(a) *Prove that if* A *and* B *are both* 2×2 *matrices with entries from* R *then* $\det(AB) = \det(A)\det(B)$.

(b) *Prove that* A *has a matrix inverse with entries from* R *if and only if* $\det(A)$ *has an inverse in* R.

(c) *Prove that when* R *is a field,* A *has a matrix inverse with entries from* R *if and only if* $\det(A) \neq 0$.

A.2. Polynomial Algebra over Fields

A.2.1. Polynomial rings over fields

We have introduced fields in order to give arithmetic structure to our alphabet F. Our next wish is then to give arithmetic structure to words formed from our alphabet. Additive structure has been provided by considering words as members of the vector space F^n of n-tuples from F for appropriate n. Scalar multiplication in F^n does not however provide a comprehensive multiplication for words and vectors. In order to construct a workable definition of multiplication for words, we introduce polynomials over F.

Let F be a field, and let x be a symbol not one of those of F, an *indeterminate*. To any n-tuple

indeterminate

$$(a_0, a_1, a_2, \ldots, a_{n-1})$$

of members of F we associate the *polynomial* in x :

polynomial

$$a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_{n-1}x^{n-1}.$$

In keeping with common notation we usually write a_0 for a_0x^0 and a_1x for a_1x^1. Also we write $0 \cdot x^i = 0$ and $1 \cdot x^i = x^i$. We sometimes use summation notation for polynomials:

$$\sum_{i=0}^{d} a_i x^i = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_dx^d.$$

$F[x]$ We next define $F[x]$ as the set of all polynomials in x over F:

$$F[x] = \{\sum_{i=0}^{\infty} a_i x^i \mid a_i \in F,\ a_i = 0 \text{ for all but a finite number of } i\}.$$

Polynomials are added in the usual manner:

$$\sum_{i=0}^{\infty} a_i x^i + \sum_{i=0}^{\infty} b_i x^i = \sum_{i=0}^{\infty} (a_i + b_i) x^i.$$

This addition is compatible with vector addition of n-tuples in the sense that if the vector $\mathbf{a}$ of F^n is associated with the polynomial $a(x)$ and the vector $\mathbf{b}$ is associated with the polynomial $b(x)$, then the vector $\mathbf{a} + \mathbf{b}$ is associated with the polynomial $a(x) + b(x)$.

Polynomial multiplication is also familiar:

$$\sum_{i=0}^{\infty} a_i x^i \cdot \sum_{j=0}^{\infty} b_j x^j = \sum_{k=0}^{\infty} c_k x^k,$$

where the coefficient c_k is given by convolution: $c_k = \sum_{i+j=k} a_i b_j$. This multiplication is the inevitable consequence of the distributive law provided we require

that $ax^i \cdot bx^j = (ab)x^{i+j}$ always. (As usual we shall omit the $\cdot$ in multiplication when convenient.)

The set $F[x]$ equipped with the operations $+$ and $\cdot$ is the *polynomial ring* in x over the field F. F is the field of *coefficients* of $F[x]$. polynomial ring coefficients

Polynomial rings over fields have many of the properties enjoyed by fields. $F[x]$ is closed and distributive nearly by definition. Commutativity and additive associativity for $F[x]$ are easy consequences of the same properties for F, and multiplicative associativity is only slightly harder to check. The constant polynomials $0x^0 = 0$ and $1x^0 = 1$ serve respectively as additive and multiplicative identities. The polynomial $ax^0 = a$, for $a \in F$, is usually referred to as a *constant polynomial* or a *scalar polynomial*. Indeed if we define scalar multiplication by $\alpha \in F$ as multiplication by the scalar polynomial $\alpha(= \alpha x^0)$, then $F[x]$ with polynomial addition and this scalar multiplication is a vector space over F, a basis for which is the subset $\{1, x, x^2, x^3, \ldots, x^i, \ldots\}$. (Note that $(-1) \cdot a(x)$ is the additive inverse of the polynomial $a(x)$.) constant polynomial scalar polynomial

We therefore see that, as with the integers, the only thing that keeps $F[x]$ from being a field is the lack of multiplicative inverses. Every nonzero scalar polynomial in $F[x]$ has a multiplicative inverse, but as we shall see below no other polynomial of $F[x]$ does. Again like the integers, $F[x]$ does satisfy the cancellation law and so is an integral domain. These last two claims follow from some results of independent interest. To prove them we first need a (once again familiar) definition. The *degree* of the polynomial $a(x)$ of $F[x]$ is the largest power of x occurring in $a(x)$ with a nonzero coefficient. Thus $a(x) = \sum_{i=0}^{d} a_i x^i$ has degree d provided that a_d is not 0. In this case we write $\deg(a(x)) = d$. This defines a degree for all nonzero polynomials. By convention we define the degree of the scalar polynomial 0 to be $-\infty$. degree

Every polynomial $a(x)$ of degree d not equal to $-\infty$ has a unique scalar multiple whose x^d term has coefficient 1. (Indeed if $a(x) = \sum_{i=0}^{d} a_i x^i$, then the appropriate multiple is $a_d^{-1} a(x)$.) A polynomial whose highest degree term has coefficient 1 is called *monic*. monic

(A.2.1) PROPOSITION. *Let $a(x)$ and $b(x)$ be polynomials of $F[x]$, for F a field.*

(1) $\deg(a(x)) + \deg(b(x)) = \deg(a(x)b(x))$;

(2) $\deg(a(x) + b(x)) \leq \max\Big(\deg(a(x)), \deg(b(x))\Big)$. □

(A.2.2) PROBLEM. *Prove the proposition.*

(A.2.3) COROLLARY. *Let F be a field.*

(1) *An element of $F[x]$ has a multiplicative inverse if and only if it has degree 0.*

(2) *If $a(x) \cdot b(x) = 0$, then $a(x) = 0$ or $b(x) = 0$.*

(3) *If $a(x)$ is nonzero and $a(x)b(x) = a(x)c(x)$ in $F[x]$, then $b(x) = c(x)$.*

PROOF. (1) We have already mentioned that polynomials of degree 0 (*i.e.*, nonzero scalars) have inverses. Suppose that $b(x)$ is the inverse of the polynomial

$a(x)$. Then $a(x)b(x) = 1$, so examining degrees we have $\deg(a(x)) + \deg(b(x)) = 0$. The two terms on the left must be either $-\infty$ or nonnegative integers. We see that the only possibility is that both are 0, as claimed.

(2) The righthand side has degree $-\infty$, so at least one of the factors on the lefthand side has degree $-\infty$ as well.

(3) Apply (2) to the equation $a(x)(b(x) - c(x)) = 0$. □

(A.2.4) PROBLEM. *Let $a(x), b(x), c(x), d(x)$ be nonzero polynomials in $F[x]$, for F a field. Suppose that $a(x) = b(x)c(x)$ and $b(x) = a(x)d(x)$. Prove that $c(x) = c$ is a constant polynomial and that $d(x) = c^{-1}$ is also constant.*

A.2.2. The division algorithm and roots

In the previous section we noted that, like the integers, polynomial rings over fields are integral domains. Continuing the parallel with the integers, we note that although in general polynomials do not have inverses, we can still perform division with remainder terms.

(A.2.5) THEOREM. *(THE DIVISION ALGORITHM.) Let F be a field, and let $a(x)$ and $b(x)$ be two polynomials of $F[x]$, $b(x)$ not 0. Then there are unique polynomials $q(x)$ and $r(x)$ satisfying*
(1) $a(x) = b(x)q(x) + r(x)$;
(2) $\deg(r(x)) < \deg(b(x))$. □

This is to be compared with the standard result in the integers that for integers a, b with $b \neq 0$ there are unique q and r with $a = bq + r$ and $0 \leq r < b$. The division algorithm is absolutely essential for us, but its proof is largely mechanical, composed of a verification that standard "polynomial long-division" works, as expected. There is a subtlety here, however. In particular, the division algorithm is not valid for polynomials with coefficients from the integers rather than fields. The difficulty is not hard to see. If $a(x) = \sum_{i=0}^{m} a_i x^i$ and $b(x) = \sum_{j=0}^{n} b_j x^j$, where $m = \deg(a(x))$ is larger than $n = \deg(b(x))$, then to carry out the first step of the long-division we must subtract from $a(x)$ the multiple $(a_m/b_n)x^{m-n}b(x)$ of $b(x)$. This requires the ability to divide by b_n in F, and this for arbitrary nonzero b_n in F, if the division algorithm is to be true in general.

Of course, the most important case is when the remainder term $r(x)$ is 0. If, for nonzero polynomials $a(x)$ and $b(x)$ of $F[x]$, there is a polynomial $q(x) \in F[x]$ with $a(x) = b(x)q(x)$, then we say that $b(x)$ is a *factor* of $a(x)$, that $a(x)$ is a *multiple* of $b(x)$, and that $b(x)$ *divides* $a(x)$.

factor
multiple
divides

(A.2.6) PROBLEM. *Prove Theorem A.2.5.*
(HINT: For existence, subtract $(a_m b_n^{-1})x^{m-n}b(x)$ from $a(x)$, then use induction to divide $b(x)$ into the resulting polynomial of smaller degree than $a(x)$. For uniqueness subtract one such answer from the other to get 0, and from this conclude first that the two remainder terms are equal and then that the two dividends are equal. It is important here that $F[x]$ is an integral domain.)

For an α in F and polynomial $p(x) = \sum_i p_i x^i$ in $F[x]$, we write $p(\alpha)$ for $\sum_i p_i \alpha^i$. We call this *evaluation* of $p(x)$ at α, or, more loosely, substituting α for x in $p(x)$.

evaluation

(A.2.7) PROBLEM. *Prove that if $p(x) + q(x) = a(x)$ and $p(x)q(x) = b(x)$ in $F[x]$, then $p(\alpha) + q(\alpha) = a(\alpha)$ and $p(\alpha)q(\alpha) = b(\alpha)$ in F.*

An easy consequence of the division algorithm is

(A.2.8) LEMMA. *For $p(x)$ a polynomial of $F[x]$, F a field, and $\alpha \in F$, $p(x) = (x - \alpha)q(x) + p(\alpha)$.*

PROOF. By the Division Algorithm A.2.5 there are polynomials $q(x)$ and $r(x)$ such that $p(x) = (x-\alpha)q(x) + r(x)$ with $\deg(r(x)) < \deg(x-a) = 1$. The polynomial $r(x)$ must therefore be a constant r. We find the exact value of r by substituting α for x: $p(\alpha) = (\alpha-\alpha)q(\alpha) + r = 0 + r = r$. □

Notice that a particular consequence of Lemma A.2.8 is that $p(\alpha)$ is 0 if and
root only if $x-\alpha$ is a factor of $p(x)$. If this is the case, then we say that α is a *root* of $p(x)$.

(A.2.9) LEMMA. *Let $p(x) = a(x)b(x)$ where $a(x), b(x), p(x)$ are polynomials of $F[x]$ for F a field. If $\alpha \in F$ is a root of $p(x)$, then it is a root of either $a(x)$ or $b(x)$.*

PROOF. $0 = p(\alpha) = a(\alpha)b(\alpha)$. As F is a field, this forces either $a(\alpha) = 0$ or $b(\alpha) = 0$. □

(A.2.10) PROPOSITION. *Let $p(x)$ be a nonzero polynomial in $F[x]$, F a field, of degree d. Then $p(x)$ has at most d distinct roots in F.*

PROOF. The proof proceeds by induction on d. The result is clearly true for $d = 0, 1$. Assume now that $d > 1$ and that the proposition is true for all polynomials of degree less than d. Consider a polynomial $p(x)$ of degree d. If $p(x)$ has no roots in F, then the proposition clearly holds for $p(x)$ (as $0 < d$). Thus we may assume that $p(x)$ has at least one root, α say. Then by Lemma A.2.8 $p(x) = (x-\alpha)q(x)$, for some $q(x)$ of degree $d-1$ (by Proposition A.2.1). By Lemma A.2.9 any root of $p(x)$ other than α must also be a root of $q(x)$. However by induction $q(x)$ has at most $d-1$ roots. Therefore $p(x)$ has at most $1 + (d-1) = d$ roots, as claimed. This completes the induction. □

(A.2.11) THEOREM. *(*LAGRANGE INTERPOLATION.*) Let $f(x)$ be a polynomial of degree d in $F[x]$, F a field. Assume that, for distinct $\alpha_1, \ldots, \alpha_n$ of F with $d < n$, we have $f(\alpha_i) = \beta_i$. Then*

$$f(x) = \sum_{i=1}^{n} \beta_i \left(\prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j} \right)$$

PROOF. Let $g(x)$ be the righthand side of the equation, a polynomial of degree at most $n-1$ with $g(\alpha_i) = \beta_i$, for $i = 1, \ldots, n$. Therefore $f(x) - g(x)$ has degree at most $n-1$ but has at least n distinct roots $\alpha_1, \ldots, \alpha_n$. Thus $f(x) - g(x)$ is the zero polynomial by Proposition A.2.10. That is, $f(x) = g(x)$. □

(A.2.12) PROBLEM. *Let $a_0, \ldots, a_m, b_0, ..., b_m$ be elements of the field F with the a_i nonzero. Then the columns of the matrix*

$$P = \begin{bmatrix} a_0 b_0^0 & a_1 b_1^0 & \cdots & a_m b_m^0 \\ a_0 b_0^1 & a_1 b_1^1 & \cdots & a_m b_m^1 \\ \vdots & \vdots & \ddots & \vdots \\ a_0 b_0^m & a_1 b_1^m & \cdots & a_m b_m^m \end{bmatrix}$$

*are linearly independent over F if and only if the b_j are distinct. (*HINT: *By Proposition A.1.12 the columns of square P are linearly independent if and only if its rows are linearly independent. Prove that a linear dependence of the rows of P corresponds to a polynomial that has each b_j as a root.)*

REMARK. If in P we choose $a_i = 1$, for all i, the result is the usual matrix of Vandermonde type. Then Problem A.2.12 asserts the well-known fact that the determinant of a Vandermonde matrix is nonzero. The matrix P can also be viewed as a generalization of the usual discrete Fourier transform matrix.

A.2.3. Modular polynomial arithmetic

Starting with the infinite integral domain $\mathbb{Z}$ we found finite rings by doing arithmetic modulo specific fixed integers. This gave us finite alphabets with good arithmetic structure. We would now like to extend this by giving structure to strings of alphabet members. This is achieved by doing arithmetic in the integral domain $F[x]$, F a field, modulo a specific fixed polynomial.

Let d be any positive integer. For any field F, let $F[x]_d$ be the set of all polynomials of $F[x]$ of degree less than d, that is,

$$F[x]_d = \{f_0 + f_1 x + f_2 x^2 + \cdots + f_{d-1}x^{d-1} \mid f_0, f_1, f_2, \ldots, f_{d-1} \in F\}.$$

Then with the usual scalar multiplication and polynomial addition $F[x]_d$ is a vector space over F of dimension d. Can we define a multiplication on $F[x]_d$ to make it into a ring? Using the division algorithm we can (in fact in several different ways).

Let $m(x)$ be a fixed polynomial of $F[x]$ of degree d. By the Division Algorithm A.2.5, for any polynomial $p(x)$ there is a unique polynomial $r(x)$ determined by:

(i) $\deg(r(x)) < d$;

(ii) $p(x) - r(x)$ is a multiple of $m(x)$ in $F[x]$.

Now we may define a multiplication on $F[x]_d$. For $a(x), b(x)$ in $F[x]_d$ we define multiplication modulo $m(x)$ by

$$a(x) \cdot b(x) = r(x) \pmod{m(x)}$$

where $r(x)$ is the remainder of $a(x)b(x)$ upon division by $m(x)$. We write $F[x] \pmod{m(x)}$ for the set $F[x]_d$ equipped with the operations of usual polynomial addition and multiplication modulo the polynomial $m(x)$ of degree d. It is now routine to check that these operations satisfy the first six axioms of Section A.1.1, giving:

$F[x] \pmod{m(x)}$

(A.2.13) LEMMA. *For any nonconstant polynomial $m(x)$, $F[x] \pmod{m(x)}$ is a commutative ring.* □

It should be noted that Lemma A.2.13 is not a purely trivial observation, but its subtlety is largely embedded in the original definition. The least obvious fact is that we are able to define multiplication consistently. The division algorithm is required to do that. Checking multiplicative associativity and distributivity also requires some care.

For the integers we found that modular arithmetic produced a field (indeed an integral domain) precisely when the modulus was a prime. What is the counterpart to a prime for polynomial rings? A polynomial $m(x) \in F[x]$ of degree d (> 0) is a *prime polynomial* if whenever $m(x)$ divides the product $a(x)b(x)$ of the two polynomials $a(x)$ and $b(x)$, then in fact it divides at least one of $a(x)$ and $b(x)$.

prime polynomial

The following theorem is the counterpart for polynomial rings over fields of the earlier result Theorem A.1.1 for the integers.

(A.2.14) THEOREM. *Let F be a finite field and $m(x)$ a nonconstant polynomial of $F[x]$. The following are equivalent:*

(1) *$m(x)$ is prime;*
(2) *$F[x] \pmod{m(x)}$ is an integral domain;*
(3) *$F[x] \pmod{m(x)}$ is a field.*

PROOF. *(3) implies (2)* by the definitions, and *(2) implies (3)* by Exercise A.1.2, since the integral domain under discussion is finite with $|F|^{\deg(m(x))}$ elements.

(2) implies (1): If $m(x)$ divides the product $a(x)b(x)$, then it must also divide the product $a_1(x)b_1(x)$, where $a_1(x)$ is the remainder of $a(x)$ upon division by $m(x)$ and $b_1(x)$ is the remainder of $b(x)$ upon division by $m(x)$. Here both $a_1(x)$ and $b_1(x)$ have smaller degree than $m(x)$. But then we have $a_1(x) \cdot b_1(x) = 0 \pmod{m(x)}$, so either $a_1(x) = 0 \pmod{m(x)}$ or $b_1(x) = 0 \pmod{m(x)}$ in the integral domain $F[x] \pmod{m(x)}$. One of the remainders is 0, and $m(x)$ divides $a(x)$ or $b(x)$.

(1) implies (2): Let $g(x)$ and $h(x)$ be members of $F[x] \pmod{m(x)}$, that is, polynomials of degree less than that of the prime polynomial $m(x)$. If $g(x)h(x) = 0 \pmod{m(x)}$, then the product $g(x)h(x)$ is a multiple of $m(x)$. Since $m(x)$ is prime, either $g(x)$ or $h(x)$ is a multiple of $m(x)$. That is, $g(x) = 0 \pmod{m(x)}$ or $h(x) = 0 \pmod{m(x)}$.

The theorem is in fact true without the assumption that $|F|$ is finite, a fact used here only in the proof that (2) implies (3). A strengthened version of the theorem will appear later as Theorem A.2.22.

Related to the modular arithmetic statement

$$p(x) = q(x) \pmod{m(x)} \quad \text{in} \quad F[x] \pmod{m(x)}$$

is the *modular congruence* statement

modular congruence

$$p(x) \equiv q(x) \pmod{m(x)} \quad \text{in} \quad F[x]\,,$$

which, by definition, says that

$$p(x) - q(x) = f(x)m(x)\,,$$

for some polynomial $f(x) \in F[x]$. That is, $p(x)$ and $q(x)$ are congruent modulo $m(x)$ precisely when their difference is a multiple of $m(x)$. Equivalently, $p(x)$ and $q(x)$ have the same remainder upon division by $m(x)$.

We are familiar with congruence statements over the integers, where, for instance, the even integers are precisely those integers congruent to 0 modulo 2 and the odd integers consist of those integers congruent to 1 modulo 2. Arithmetic in the integers modulo 2, $\mathbb{Z}_2$, can be interpreted as making statements about the relationship between the set of even integers (represented by 0) and the set of odd integers (represented by 1). For instance, we have

"The sum of an odd integer and an even integer is odd."
"The product of two odd integers is odd."

Similar statements hold for arithmetic done modulo any integer:

> "The product of two numbers each 2 more than a multiple of 3
> is a number that is 1 more than a multiple of 3."

That is, the product of two numbers, each congruent to 2 modulo 3 is a number congruent to 1 modulo 3.

Polynomial modular arithmetic has a similar relationship to polynomial modular congruences, as the following problem demonstrates.

(A.2.15) PROBLEM.
Prove that, if $a_1(x) \equiv a(x) \pmod{m(x)}$ *and* $b_1(x) \equiv b(x) \pmod{m(x)}$, *then*
(*a*) $a_1(x) + b_1(x) \equiv a(x) + b(x) \pmod{m(x)}$; *and*
(*b*) $a_1(x)b_1(x) \equiv a(x)b(x) \pmod{m(x)}$.

We see that the statement

$$a(x) + b(x) = c(x) \pmod{m(x)}$$

is a special case of

$$a(x) + b(x) \equiv c(x) \pmod{m(x)}\,;$$

and, similarly,

$$a(x)b(x) = c(x) \pmod{m(x)}$$

is a special case of

$$a(x)b(x) \equiv c(x) \pmod{m(x)}\,.$$

With this in mind, we will usually use the symbol $=$ in place of $\equiv$, abusing our notation by writing

$$a(x)b(x) = c(x) \pmod{m(x)}$$

even when the polynomials $a(x)$, $b(x)$, and $c(x)$ have degree larger than that of $m(x)$.

A.2.4. Greatest common divisors and unique factorization

We introduce the important concept of the greatest common divisor of a pair (or set) of polynomials.

(A.2.16) THEOREM. *In $F[x]$, F a field, let $a(x)$ and $b(x)$ be two polynomials not both equal to 0. Then there is a unique monic polynomial $g(x)$ in $F[x]$ such that:*

(i) $a(x)$ and $b(x)$ are multiples of $g(x)$;

(ii) if $n(x)$ divides both $a(x)$ and $b(x)$ then $g(x)$ is a multiple of $n(x)$.

Indeed $g(x)$ is the unique monic polynomial of minimal degree in the set

$$G = \{\, s(x)a(x) + t(x)b(x) \mid s(x), t(x) \in F[x] \}\,.$$

PROOF. Choose in the set G a monic polynomial $g(x) = s(x)a(x)+t(x)b(x)$ of smallest degree. This determines $g(x)$ uniquely, since if $g^*(x) = s^*(x)a(x) + t^*(x)b(x)$ were a different monic polynomial in G of the same degree as $g(x)$, then

$$g(x) - g^*(x) = \big(s(x) - s^*(x)\big)a(x) + \big(t(x) - t^*(x)\big)b(x)$$

would have a monic multiple of smaller degree that still belonged to G.

If $n(x)$ divides $a(x)$ and $b(x)$, then it certainly divides $g(x) = s(x)a(x) + t(x)b(x)$, giving (ii).

It remains to check (i). By the Division Algorithm A.2.5 there are polynomials $q(x)$ and $r(x)$ with $a(x) = q(x)g(x) + r(x)$ and $\deg(r(x)) < \deg(g(x))$. Here

$$\begin{aligned} r(x) &= a(x) - q(x)g(x) \\ &= 1 \cdot a(x) - q(x)(s(x)a(x) + t(x)b(x)) \\ &= \big(1 - q(x)s(x)\big)\, a(x) + \big(-q(x)t(x)\big)\, b(x)\,. \end{aligned}$$

Therefore $r(x)$ is a member of G with smaller degree than that of $g(x)$. Our choice of $g(x)$ now forces $r(x) = 0$. Thus $a(x) = q(x)g(x)$ is a multiple of $g(x)$, as desired. Similarly, $b(x)$ is a multiple of $g(x)$, giving (i). □

The polynomial $g(x)$ of Theorem A.2.16 is called the *greatest common divisor* of $a(x)$ and $b(x)$. In this case we often write $g(x) = \gcd(a(x), b(x))$. Notice that, for nonzero $a(x)$, $\gcd(a(x), 0)$ is the unique monic scalar multiple of $a(x)$. If $a(x) = b(x) = 0$, then $g(x) = 0$ satisfies (1) and (2) trivially, so we set $\gcd(0, 0) = 0$ (even though it is not monic). greatest common divisor

If $\gcd(a(x), b(x)) = 1$, we say that $a(x)$ and $b(x)$ are *relatively prime*. We have immediately relatively prime

(A.2.17) COROLLARY. *The polynomials $a(x), b(x) \in F[x]$ are relatively prime if and only if there are $s(x), t(x) \in F[x]$ with $s(x)a(x) + t(x)b(x) = 1$.* □

The corollary implies that

$$s(x)a(x) = 1 \pmod{b(x)}\,,$$

that is, $s(x)$ is the multiplicative inverse of $a(x)$ modulo $b(x)$; and we sometimes then even write

$$s(x) = a(x)^{-1} = \frac{1}{a(x)} \pmod{b(x)}\,.$$

To repeat, if $a(x)$ and $b(x)$ are relatively prime, then we can invert $a(x)$ modulo $b(x)$. Indeed $a(x)$ is invertible modulo $b(x)$ if and only if $a(x)$ and $b(x)$ are relatively prime.

An argument similar to that of Theorem A.2.16 gives the more general and fundamental result

(A.2.18) THEOREM. *Let F be a field and*

$$S = \{f_i(x) \,|\, i \in I\}$$

a (possibly infinite) set of polynomials in $F[x]$, not all equal to 0. Consider the set

$$G = \left\{ \sum_{i \in I} a_i f_i(x) \,|\, a_i \in F \right\}$$

Here, when the index set I is infinite, it should be understood that all but a finite number of the a_i must be 0 in any particular sum. Then

(1) *G contains a unique monic polynomial $g(x)$ of minimal degree;*

(2) *$g(x)$ has the two properties:*

(i) every member of S is a multiple of $g(x)$;

(ii) if $n(x)$ divides every member of S then $g(x)$ is a multiple of $n(x)$. □

greatest common divisor

The polynomial $g(x)$ of Theorem A.2.18 is called the *greatest common divisor* of the set of polynomials S, and we write $g(x) = \gcd(S)$. By convention, the gcd of any empty set of polynomials is 1, and again $\gcd(\{0\}) = 0$.

(A.2.19) PROBLEM. *Let F be a field and*

$$S = \{f_1(x), \ldots, f_i(x), \ldots, f_m(x)\}$$

a finite set of polynomials in $F[x]$, not all equal to 0.

Set

$$L = \{f(x) \,|\, f(x) \text{ is a multiple of } f_i(x), \text{ for } i = 1, \ldots, m\}\,,$$

and let $l(x)$ be the greatest common divisor of the set L. Prove that $l(x)$ has the two properties:

(i) $l(x)$ is a multiple of $f_i(x)$ for $i = 1, \ldots, n$;

(ii) if $n(x)$ is a multiple of $f_i(x)$ for $i = 1, \ldots, n$, then $n(x)$ is a multiple of $l(x)$.

least common multiple

This polynomial $l(x)$ is called the least common multiple *of S, written* $\operatorname{lcm}(S)$.

(A.2.20) LEMMA. *Let F be a field, and let $p(x), q(x), m(x) \in F[x]$. Suppose $m(x)$ divides the product $p(x)q(x)$ but $m(x)$ and $p(x)$ are relatively prime,* $\gcd(m(x), p(x)) = 1$. *Then $m(x)$ divides $q(x)$.*

PROOF. By Corollary A.2.17 there are polynomials $s(x)$ and $t(x)$ such that $1 = s(x)m(x) + t(x)p(x)$. Therefore

$$\begin{aligned} q(x) &= 1 \cdot q(x) \\ &= \Big(s(x)m(x) + t(x)p(x)\Big)q(x) \\ &= \Big(s(x)q(x)\Big)m(x) + t(x)\Big(p(x)q(x)\Big). \end{aligned}$$

Here $m(x)$ divides both terms of the righthand side, so $m(x)$ divides $q(x)$. □

Let $m(x)$ be a polynomial in $F[x]$ of degree $d > 0$. Then $m(x)$ is *reducible* in $F[x]$ if there are polynomials $a(x) \in F[x]$ with $0 < \deg(a(x)) < d$ and $b(x) \in F[x]$ with $0 < \deg(b(x)) < d$ such that $m(x) = a(x)b(x)$. That is, $m(x)$ is reducible if it can be written as a product of polynomials of smaller degree. Otherwise $m(x)$ is *irreducible*. (Constant polynomials are neither reducible nor irreducible.) If irreducible $m(x) = a(x)b(x)$, then one of the factors is a nonzero constant, say $a(x) = a$, and the other factor $b(x) = a^{-1}m(x)$ has the same degree as $m(x)$.

reducible

irreducible

Recall that in Section A.2.3 we defined $m(x)$ to be prime if whenever $m(x)$ divides the product $a(x)b(x)$, it must in fact divide one of $a(x)$ and $b(x)$. If prime $m(x) = a(x)b(x)$, then $m(x)$ must divide either $a(x)$ or $b(x)$; so they can not both have degree less than that of $m(x)$. Thus every prime is irreducible. We use Lemma A.2.20 to prove the converse.

(A.2.21) LEMMA. *In $F[x]$ with F a field, every irreducible polynomial is prime.*

PROOF. Suppose irreducible $m(x)$ divides the product $a(x)b(x)$. If we have $\gcd(m(x), a(x)) = 1$, then by Lemma A.2.20 the polynomial $b(x)$ is divisible by $m(x)$, as required. So we may assume that $m(x)$ and $a(x)$ are not relatively prime. As $m(x)$ is irreducible, its divisor $\gcd(m(x), a(x)) \neq 1$ must therefore be $c \cdot m(x)$, for some constant c. In particular, $m(x) = c^{-1}\gcd(m(x), a(x))$ divides $a(x)$. □

We are now in a position to give the strengthened version of Theorem A.2.14.

(A.2.22) THEOREM. *Let F be a field and $m(x)$ a nonconstant polynomial of $F[x]$. The following are equivalent:*

(1) *$m(x)$ is irreducible;*

(2) *$F[x] \pmod{m(x)}$ is an integral domain;*

(3) *$F[x] \pmod{m(x)}$ is a field.*

PROOF. By Lemma A.2.21 the present (1) is equivalent to that of Theorem A.2.14.

In Theorem A.2.14 we assumed that the field F was finite; but, as noted there, the assumption was only used in the proof that (2) implies (3). In particular from Theorem A.2.14 we know that (3) implies (2) and (2) implies (1).

We now prove that (1) implies (3), giving the theorem. Let $f(x) \in F[x]$ of degree less than that of $m(x)$. This implies that $f(x)$ and irreducible $m(x)$ are relatively prime. There exist polynomials $s(x)$ and $t(x)$ with $f(x)s(x) + m(x)t(x) = 1$. Therefore $f(x)s(x) = 1 \pmod{m(x)}$. If $g(x)$ is the remainder of $s(x)$ upon division by $m(x)$, then again $f(x)g(x) = 1$ in $F[x] \pmod{m(x)}$. That is, $f(x)$ is invertible in $F[x] \pmod{m(x)}$, as required for (3). □

In the integers, we have unique factorization into primes. To be precise, every nonzero integer is plus or minus a product of positive prime numbers, this factorization being unique up to the order in which the primes appear. Essentially the same result is true for polynomial rings over fields.

(A.2.23) THEOREM. *(*UNIQUE FACTORIZATION OF POLYNOMIALS.*) Let F be a field, and $f(x)$ a nonzero member of $F[x]$. Then $f(x)$ can be written as a product $f(x) = c\prod_{i=1}^{n} f_i(x)$ of a nonzero constant c and a collection of monic irreducible polynomials $f_i(x)$. This factorization is unique up to the order in which the irreducibles $f_i(x)$ are taken.* □

We sketch a proof with the details left to Problem A.2.24. The existence of factorizations into irreducibles is easy to see, as is the uniqueness of factorization into primes. Since, by Lemma A.2.21, in this situation all irreducibles are primes, the result follows.

(A.2.24) PROBLEM. *Prove Theorem A.2.23.*

*(*HINT: *Deal first with existence. Every nonzero polynomial $f(x)$ is a product $cg(x)$ with c a nonzero constant and $g(x)$ monic, so assume that $f(x)$ is monic. If $f(x)$ is not irreducible, then it can be factored as a product $g_1(x)g_2(x)$ of two monic polynomials of smaller degree. Either they are irreducible or they can be split further as products. Proceed in this fashion; use induction. As the degrees decrease at each stage, this process must stop with $f(x)$ written as a product of irreducible polynomials.*

Now consider uniqueness. Let $f(x) = d\prod_{j=1}^{m} g_j(x)$ be a second factorization into monic irreducibles. Then $c = d$ is the coefficient of the highest degree term. The monic irreducible $f_1(x)$ divides the product $g_1(x)(\prod_{j=2}^{m} g_j(x))$. By Lemma A.2.21 irreducibles are prime, so either $f_1(x) = g_1(x)$ or $f_1(x)$ divides $\prod_{j=2}^{m} g_j(x)$. In the second case $f_1(x)$ divides $\prod_{j=2}^{m} g_j(x) = g_2(x)(\prod_{j=3}^{m} g_j(x))$; so as before either $f_1(x)$ equals $g_2(x)$ or it divides $\prod_{j=3}^{m} g_j(x)$. Proceeding in this fashion, $f_1(x)$ is equal to one of the irreducible monic factors $g_j(x)$; use induction again. A similar argument shows that $f_2(x)$ is also equal to one of the $g_j(x)$, and indeed that each $f_i(x)$ is equal to one of the $g_j(x)$. Compare degrees to conclude finally that $n = m$.)

(A.2.25) PROBLEM. *Let*

$$S = \{f_1(x), ..., f_i(x), ..., f_m(x)\}$$

be a finite set of polynomials, as in Problem A.2.19 above. Suppose there are constants c_i, distinct monic irreducible polynomials $p_j(x)$, and nonnegative integers $e_{i,j}$, $1 \le j \le n$, such that, for each i,

$$f_i(x) = c_i \prod_{j=1}^{n} p_j(x)^{e_{i,j}} \ .$$

For each j, let $d_j = \max_i(e_{i,j})$. Prove that $\mathrm{lcm}(S) = \prod_{j=1}^{n} p_j(x)^{d_j}$.

(A.2.26) PROBLEM. *For the polynomial* $p(x) = \sum_{i=0}^{k} p_i x^i \in F[x]$, *define the* formal derivative *of* $p(x)$, *denoted* $p'(x)$, *by* $p'(x) = \sum_{i=1}^{k} i p_i x^{i-1}$. *Prove the usual product rule for derivatives:* $(a(x)b(x))' = a(x)b'(x) + a'(x)b(x)$. formal derivative

(A.2.27) PROBLEM. *Consider the polynomial ring* $F[x]$, F *a field; and let* $\alpha \in F$ *be a root of* $p(x) \in F[x]$. *Prove that* $(x-\alpha)^2$ *divides* $p(x)$ *if and only if* $x - \alpha$ *divides the formal derivative* $p'(x)$.

(A.2.28) PROBLEM. *A polynomial* $f(x)$ *is* square free *in* $F[x]$ *if there are no nonconstant polynomials* $g(x) \in F[x]$ *for which* $g(x)^2$ *divides* $f(x)$. *Prove that in* $F[x]$, *the polynomial* $f(x)$ *is square free if and only if we have* $\gcd(f(x), f'(x)) = 1$. square free

In particular, over $\mathbb{F}_2$ *all derivatives are squares.*

A.3. Special Topics

A.3.1. The Euclidean algorithm

Let F be a field. In Theorem A.2.16 we gave a nonconstructive proof for the existence of the greatest common divisor of two polynomials $a(x)$ and $b(x)$ of $F[x]$. The Euclidean algorithm is an algorithm that constructs $\gcd(a(x), b(x))$ explicitly. The basic method is simple. If $q(x)$ is any polynomial, then

$$\gcd(a(x), b(x)) = \gcd(a(x) - q(x)b(x), b(x)).$$

In particular, $a(x)$ can be replaced in the calculation by its remainder $r(x)$ upon division by $b(x)$. Assuming that $a(x)$ has degree at least as big as that of $b(x)$, the remainder $r(x)$ will have smaller degree than $a(x)$; so the gcd of the original pair of polynomials will be equal to the gcd of a new pair with smaller total degree. We can continue in this fashion decreasing the degree of the remainder at each stage until the process stops with remainder 0, and at this point the gcd becomes clear.

In fact the approach we take is a little different. From our proof of Theorem A.2.16 we know that $\gcd(a(x), b(x))$ is the monic polynomial of minimal degree within the set

$$G = \{\, s(x)a(x) + t(x)b(x) \mid s(x), t(x) \in F[x] \,\}$$

Thus we examine all equations of the form

$$p(x) = s(x)a(x) + t(x)b(x)\,,$$

looking for one in which nonzero $p(x)$ has minimal degree. The unique monic scalar multiple of this $p(x)$ is then equal to $\gcd(a(x), b(x))$.

If we have two suitable equations:

$$m(x) = e(x)a(x) + f(x)b(x)\,; \tag{A.1}$$
$$n(x) = g(x)a(x) + h(x)b(x)\,; \tag{A.2}$$

then we can find a third with lefthand side of smaller degree. Assume that the degree of $m(x)$ is at least as big as that of $n(x)$. By the Division Algorithm A.2.5 there are $q(x)$ and $r(x)$ with $m(x) = q(x)n(x)+r(x)$and $\deg(r(x)) < \deg(n(x))$. Subtracting $q(x)$ times equation (2) from equation (1) we have the desired

$$r(x) = m(x) - q(x)n(x) = \tag{A.3}$$
$$\Big(e(x) - q(x)g(x)\Big)a(x) + \Big(f(x) - q(x)h(x)\Big)b(x).$$

Next we may divide $r(x)$ into $n(x)$ and, using equations (2) and (3), further reduce the degree of the lefthand side. Continuing as before we must ultimately arrive at an equation with 0 on the left. The lefthand side of the previous equation will then have the desired minimal degree. A benefit of this method of

calculation is that the appropriate polynomials $s(x)$ and $t(x)$ are produced at the same time as the gcd.

To succeed with this approach we must have two equations to begin with. These are provided by:

$$a(x) = 1 \cdot a(x) + 0 \cdot b(x); \tag{A.4}$$
$$b(x) = 0 \cdot a(x) + 1 \cdot b(x). \tag{A.5}$$

(A.3.1) THEOREM. *(THE EUCLIDEAN ALGORITHM.)*
Assume that $\deg(a(x)) \geq \deg(b(x))$ *with* $a(x) \neq 0$. *At Step i we construct the equation*

$$\mathbf{E_i} : r_i(x) = s_i(x)a(x) + t_i(x)b(x).$$

Equation $\mathbf{E_i}$ *is constructed from* $\mathbf{E_{i-1}}$ *and* $\mathbf{E_{i-2}}$, *the appropriate initialization being provided by (4) and (5):*

$$r_{-1}(x) = a(x); \quad s_{-1}(x) = 1; \quad t_{-1}(x) = 0;$$
$$r_0(x) = b(x); \quad s_0(x) = 0; \quad t_0(x) = 1.$$

Step i. Starting with $r_{i-2}(x)$ *and* $r_{i-1}(x)$ $(\neq 0)$ *use the Division Algorithm A.2.5 to define* $q_i(x)$ *and* $r_i(x)$*:*

$$r_{i-2}(x) = q_i(x)r_{i-1}(x) + r_i(x) \text{ with } \deg(r_i(x)) < \deg(r_{i-1}(x)).$$

Next define $s_i(x)$ *and* $t_i(x)$ *by:*

$$s_i(x) = s_{i-2}(x) - q_i(x)s_{i-1}(x);$$
$$t_i(x) = t_{i-2}(x) - q_i(x)t_{i-1}(x).$$

We then have the equation

$$\mathbf{E_i} : r_i(x) = s_i(x)a(x) + t_i(x)b(x).$$

Begin with $i = 0$. *If we have* $r_i(x) \neq 0$, *then proceed to Step i+1. Eventually there will be an i with* $r_i(x) = 0$. *At that point halt and declare* $\gcd(a(x), b(x))$ *to be the unique monic scalar multiple of the nonzero polynomial* $r_{i-1}(x)$.

PROOF. For each i, $r_i(x) = r_{i-2}(x) - q_i(x)r_{i-1}(x)$; so $\mathbf{E_i}$ holds. This also shows that

$$\gcd(r_{i-1}(x), r_i(x)) = \gcd(r_{i-2}(x), r_{i-1}(x))$$
$$= \cdots = \gcd(r_{-1}(x), r_0(x)) = \gcd(a(x), b(x)).$$

As long as $i \geq 0$ and $r_i(x) \neq 0$, $\deg(r_{i+1}(x)) < \deg(r_i(x))$. Thus in at most $\deg(b(x))$ steps $r_i(x) = 0$ is reached. Then $\gcd(r_{i-1}(x), 0) = \gcd(a(x), b(x))$ is the unique monic multiple of $r_{i-1}(x)$, completing verification of the algorithm. □

(A.3.2) PROBLEM.
(a) *Prove that $q_i(x)$ of Theorem A.3.1 has positive degree, for all $i \geq 2$.*
(b) *Prove that* $\deg(s_i(x))$ *and* $\deg(t_i(x))$ *are increasing functions of $i \geq 1$.*

We can think of the Euclidean algorithm as finding a new equation $\mathbf{E_i}$ from the previous two via

$$\mathbf{E_i} = -q_i(x)\mathbf{E_{i-1}} + \mathbf{E_{i-2}}\,.$$

This provides the entry to another presentation of the Euclidean algorithm that for certain purposes is quite helpful.

Consider the matrix with entries from $F[x]$

$$R_0 = \begin{bmatrix} a(x) & 1 & 0 \\ b(x) & 0 & 1 \end{bmatrix}\,.$$

We wish, by elementary row operations over $F[x]$, to reduce this matrix to echelon form

$$R = \begin{bmatrix} p(x) & * & * \\ 0 & * & * \end{bmatrix}\,,$$

where in fact $p(x) = \gcd(a(x), b(x))$. For each $i > 1$, set

$$Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -q_i(x) \\ 0 & 1 \end{bmatrix}\,,$$

a product of the matrices for two elementary row operations. Then after defining

$$R_i = \begin{bmatrix} r_{i-1}(x) & s_{i-1}(x) & t_{i-1}(x) \\ r_i(x) & s_i(x) & t_i(x) \end{bmatrix}\,,$$

we find that $R_i = Q_i R_{i-1}$, for all $i \geq 1$. Therefore left multiplication by Q_i can be thought of as accomplishing *Step i* of the Euclidean algorithm. Because $(1, -a(x), -b(x))^\top$ is a null vector of R_0, it is also a null vector of each R_i. That is, for each i we have the equation

$$\mathbf{E_i} : r_i(x) = s_i(x)a(x) + t_i(x)b(x).$$

When first $r_i(x) = 0$, then $r_{i-1}(x)$ is a scalar multiple of $\gcd(a(x), b(x))$; so the desired matrix R can be realized as a scalar multiple of R_i.

For each $i \geq 1$, set $S_i = \prod_{j=1}^{i} Q_j$, so that $S_i R_0 = R_i$. Each Q_j has determinant equal to -1 (see Problem A.1.15), so S_i has determinant $(-1)^i$. If, for each i, we define $R_i(r,t)$ (respectively, $R_i(s,t)$) to be the 2×2 submatrix of R_i composed of the r- and t-columns (resp., s- and t-columns), then we have

$$S_i \begin{bmatrix} a(x) & 0 \\ b(x) & 1 \end{bmatrix} = S_i R_0(r,t) = R_i(r,t) = \begin{bmatrix} r_{i-1}(x) & t_{i-1}(x) \\ r_i(x) & t_i(x) \end{bmatrix}\,.$$

Similarly

$$S_i \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = S_i R_0(s,t) = R_i(s,t) = \begin{bmatrix} s_{i-1}(x) & t_{i-1}(x) \\ s_i(x) & t_i(x) \end{bmatrix}.$$

Calculating determinants, we have a proof of

(A.3.3) LEMMA. (1) $r_{i-1}(x)t_i(x) - r_i(x)t_{i-1}(x) = (-1)^i a(x)$, for $i \geq 0$.
(2) $s_{i-1}(x)t_i(x) - s_i(x)t_{i-1}(x) = (-1)^i$, for $i \geq 0$. □

(A.3.4) COROLLARY. $\gcd(s_i(x), t_i(x)) = 1$, *for all* $i \geq -1$.

PROOF. This follows from Lemma A.3.3(2) and Theorem A.2.16. □

(A.3.5) PROBLEM. *Prove that* $\deg(r_{i-1}(x)) + \deg(t_i(x)) = \deg(a(x))$, *for all* $i \geq 0$.
*(*HINT: *use Problem A.3.2(b) and Lemma A.3.3(1).)*

(A.3.6) PROBLEM.
(a) *Prove that* $r_{i-1}(x)s_i(x) - r_i(x)s_{i-1}(x) = (-1)^{i+1}b(x)$, *for all* $i \geq 0$.
(b) *Prove that* $\deg(r_{i-1}(x)) + \deg(s_i(x)) =$ *deg*$(b(x))$, *for all* $i \geq 1$.

A Euclidean Algorithm example

We calculate $\gcd(x^4, 4x^3+3x^2+5x) = x$ over $\mathbb{F}_7$ using the Euclidean algorithm. At Step i we define $q_i(x)$, $r_i(x)$, $s_i(x)$, and $t_i(x)$ using

$$\begin{aligned} r_{i-2}(x) &= q_i(x)r_{i-1}(x) + r_i(x) \\ s_i(x) &= s_{i-2}(x) - q_i(x)s_{i-1}(x) \\ t_i(x) &= t_{i-2}(x) - q_i(x)t_{i-1}(x)\ . \end{aligned}$$

Step i	$q_i(x)$	$r_i(x)$	$s_i(x)$	$t_i(x)$
-1	–	x^4	1	0
0	–	$4x^3+3x^2+5x$	0	1
1	$2x+2$	$5x^2+4x$	1	$5x+5$
2	$5x+5$	$\mathbf{6x}$	$2x+2$	$3x^2+6x+4$
3	$2x+3$	0	$3x^2+4x+2$	x^3

Step 1.

$$\begin{array}{rl} & \qquad\qquad\qquad\quad 2x \quad +2 \quad = q_1(x) \\ r_0(x) = 4x^3 \quad +3x^2 \quad +5x & \big| \; x^4 \qquad\qquad\qquad\quad = r_{-1}(x) \\ & \quad x^4 \quad +6x^3 \quad +3x^2 \\ & \qquad\quad x^3 \quad +4x^2 \\ & \qquad\quad x^3 \quad +6x^2 \quad +3x \\ & \qquad\qquad\quad 5x^2 \quad +4x \quad = r_1(x) \end{array}$$

$$\begin{aligned} r_{-1}(x) &= q_1(x)r_0(x) + r_1(x) \\ x^4 &= (2x+2)(4x^3+3x^2+5x) + (5x^2+4x) \\ q_1(x) &= 2x+2 \\ r_1(x) &= 5x^2+4x \end{aligned}$$

$$\begin{aligned} s_1(x) &= s_{-1}(x) - q_1(x)s_0(x) \\ s_1(x) &= 1 - (2x+2)0 = 1 \end{aligned}$$

$$\begin{aligned} t_1(x) &= t_{-1}(x) - q_1(x)t_0(x) \\ t_1(x) &= 0 - (2x+2)1 = 5x+5 \end{aligned}$$

Step 2.

$$\begin{aligned} r_0(x) &= q_2(x)r_1(x) + r_2(x) \\ 4x^3+3x^2+5x &= (5x+5)(5x^2+4x) + 6x \\ q_2(x) &= 5x+5 \\ r_2(x) &= 6x \end{aligned}$$

$$\begin{aligned} s_2(x) &= s_0(x) - q_2(x)s_1(x) \\ s_2(x) &= 0 - (5x+5)1 = 2x+2 \end{aligned}$$

$$\begin{aligned} t_2(x) &= t_0(x) - q_2(x)t_1(x) \\ t_2(x) &= 1 - (5x+5)(5x+5) = 3x^2+6x+4 \end{aligned}$$

Step 3.

$$\begin{aligned} r_1(x) &= q_3(x)r_2(x) + r_3(x) \\ 5x^2+4x &= (2x+3)(6x) + 0 \\ q_3(x) &= 2x+3 \\ r_3(x) &= 0 \end{aligned}$$

$$\begin{aligned} s_3(x) &= s_1(x) - q_3(x)s_2(x) \\ s_3(x) &= 1 - (2x+3)(2x+2) = 3x^2+4x+2 \end{aligned}$$

$$\begin{aligned} t_3(x) &= t_1(x) - q_3(x)t_2(x) \\ t_3(x) &= (5x+5) - (2x+3)(3x^2+6x+4) \\ &= (5x+5) - (6x^3+5x+5) = -6x^3 = x^3 \end{aligned}$$

As $r_3(x) = 0$, $\gcd(x^4, 4x^3+3x^2+5x)$ is the unique monic scalar multiple of $r_2(x) = 6x$. Thus $\mathbf{x = gcd(x^4, 4x^3+3x^2+5x)}$, as claimed.

We should also have $r_2(x) = s_2(x)x^4 + t_2(x)(4x^3+3x^2+5x)$ and therefore $x = 6r_2(x) = 6s_2(x)x^4 + 6t_2(x)(4x^3+3x^2+5x)$. We check:

$$\begin{aligned} 6r_2(x) &= 6s_2(x)x^4 + 6t_2(x)(4x^3+3x^2+5x) \\ &= 6(2x+2)x^4 + 6(3x^2+6x+4)(4x^3+3x^2+5x) \\ &= (5x+5)x^4 + (4x^2+x+3)(4x^3+3x^2+5x) \\ &= (5x^5+5x^4) + (2x^5+5x^4+6x^3) + \\ &\qquad +(4x^4+3x^3+5x^2) + (5x^3+2x^2+x) \\ &= x \text{ !!} \end{aligned}$$

A.3.2. Finite Fields

Consider a finite field F of characteristic p. (Remember from Lemma A.1.3 that this says 1 lies in a subfield of F that is a copy of $\mathbb{Z}_p = \mathbb{F}_p$.) Let α be any element of F. Any subfield (indeed any subring) of F that contains both the subfield $\mathbb{F}_p$ and α must contain the set E of all polynomials in α with coefficients in $\mathbb{F}_p$:

$$E = \{a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_k\alpha^k \mid a_i \in \mathbb{F}_p, k > 0\}.$$

Notice however that in this instance α is not an indeterminate; there are going to be various different polynomials $f(x)$ in $\mathbb{F}_p[x]$ that represent the same element $f(\alpha)$ of F. Indeed as F is finite while $\mathbb{F}_p[x]$ is infinite, this must be the case. As in the proof of Lemma A.1.3 this forces the set

$$I = \{ \text{ all polynomials } f(x) \in \mathbb{F}_p[x] \text{ with } f(\alpha) = 0 \ \}$$

to contain polynomials other than the constant polynomial 0. As in Theorem A.2.18, the greatest common divisor of the set I, $m(x) = \gcd(I)$, is called the *minimal polynomial* of α over $\mathbb{F}_p$ and is usually denoted $m_\alpha(x)$ (but also sometimes $m_{\alpha,\mathbb{F}_p}(x)$). The set I then consists of all members of $F[x]$ that are multiples of $m_\alpha(x)$. That is, the polynomial $m_\alpha(x)$ is uniquely determined in $\mathbb{F}_p[x]$ as a monic polynomial with α as a root that divides all polynomials with α as a root. We observe that a minimal polynomial must always be irreducible. Indeed if $m(x) = f(x)g(x)$, then $0 = m(\alpha) = f(\alpha)g(\alpha)$ whence $f(\alpha) = 0$ or $g(\alpha) = 0$. Therefore at least one of $f(x)$ and $g(x)$ is in I, but the greatest common divisor $m(x)$ of I has minimal degree among the nonzero elements of I.

minimal polynomial

Let us now examine the set E. E is closed under addition and multiplication and contains 0 and 1. Thus E is at least a subring of F. Furthermore no two nonzero members of E have product 0, as this is true in F itself. Thus E is moreover a sub-integral domain of F. Now Problem A.1.2 shows that E is in fact a subfield of F, indeed the smallest subfield of F that contains α. (All subfields contain 1 and so all of $\mathbb{F}_p$.) What is the arithmetic of the subfield E?

Let us assume that the minimal polynomial $m(x)$ has degree d (greater than 0). Then by the division algorithm every polynomial $f(x)$ of $\mathbb{F}_p[x]$ has a unique remainder $r(x)$ of degree less than d upon division by $m(x)$, and $f(\alpha) = r(\alpha)$ as $m(\alpha) = 0$. Thus in fact

$$E = \{\, r(\alpha) \,|\, r(x) \in \mathbb{F}_p[x] \text{ of degree } < d \,\}.$$

Furthermore two distinct polynomials $r_1(x), r_2(x) \in \mathbb{F}_p[x]_d$ can not have $r_1(\alpha) = r_2(\alpha)$, because their difference would then be a nonzero polynomial of degree less than d having α as a root. Such a polynomial would belong to I, whereas $m(x)$ has minimal degree among all nonzero members of I. In particular E has exactly p^d elements. Note also that for polynomials $a(x), b(x) \in \mathbb{F}_p[x]$ we have in E that $a(\alpha)b(\alpha) = r(\alpha)$, where $r(x)$ is the remainder of $a(x)b(x)$ upon division by $m(x)$. Thus the arithmetic of E is exactly that of $\mathbb{F}_p[x] \pmod{m(x)}$. Indeed we have:

(A.3.7) LEMMA. *Let F be a finite field of characteristic p, and let α be an arbitrary element of F. Then the smallest subfield E of F that contains α is a copy of the field $\mathbb{F}_p[x] \pmod{m_\alpha(x)}$ where $m_\alpha(x)$ is the minimal polynomial of α over $\mathbb{F}_p$.* □

We next examine a result of great theoretical and practical importance.

(A.3.8) THEOREM. *Let F be a finite field with $|F| = q$. Then there is an element α in F with the property that*

$$F - \{0\} = \{\, \alpha, \alpha^2, \ldots, \alpha^{q-2}, \alpha^{q-1} = \alpha^0 = 1\}.$$

PROOF. We first observe that for any nonzero α of F, the set

$$X = \{\, \alpha, \alpha^2, \ldots, \alpha^i, \ldots \mid i \in \mathbb{Z}^+\}$$

is finite and contained within $F - \{0\}$. As before this implies that, for each nonzero α of F, there is a positive integer n (depending upon α) with $\alpha^n = 1$. The smallest such positive n is called the *order* of α. Among all the nonzero elements of F choose α one of maximal order n, say. Note that the statement that α has order n is equivalent to the statement that the set X contains exactly n elements of F. Additionally for each $\beta = \alpha^i$ of X we have $\beta^n = (\alpha^i)^n = (\alpha^n)^i = 1^i = 1$. The crucial point in the proof is that X, for our choice of α, is precisely the set of all roots in F of the polynomial $x^n - 1$. In particular any element of F with order dividing n must belong to X. An element $\alpha \in F$ is called a *primitive n^{th} root of unity* if it has order n.

order

primitive n^{th} root of unity

Assume now that it is possible to find a nonzero element γ of F that does not belong to X. By the remark at the end of the previous paragraph the order m of g is not a divisor of n. Thus there is a prime s and a prime power s^i that divides m but does not divide n. Let $m = s^i u$ and $n = s^j v$, where i is larger than j and neither u nor v are multiples of s. A somewhat lengthy calculation suffices to check (do it!) that the element $\delta = \alpha^{s^j} \cdot \gamma^u$ has order $s^i v$. As this is larger than n we have contradicted our original choice of α. Therefore no such element γ can be found; and X is all of F, proving the theorem. □

Of course for an α as in Theorem A.3.8, F itself is the smallest subfield of F containing α. Thus from Lemma A.3.7 and Theorem A.3.8 we have:

(A.3.9) THEOREM. *Every finite field F can be written as $\mathbb{F}_p[x] \pmod{m(x)}$ for some prime p and some irreducible polynomial $m(x)$ in $\mathbb{F}_p[x]$.* □

Note that Theorem A.3.9 can be thought of as a converse to Theorem A.2.14 for finite fields.

An α as in Theorem A.3.8 is a primitive $(|F| - 1)^{\text{th}}$ root of unity in F and is called a *primitive element* of F. Its minimal polynomial is called a *primitive polynomial*. Thus Theorem A.3.9 remains true with the word 'primitive' in place of 'irreducible'.

primitive element

primitive polynomial

One consequence of Theorem A.3.9 is that a finite field must have the number of its elements equal to a power of a prime (although we already knew this from Problem A.1.6). By Lemma A.1.3 there are fields of prime order for every prime, but what about every prime power? For the time being we are content to state without proof:

(A.3.10) THEOREM. *For each prime p and each positive integer d, there exist fields containing exactly p^d elements.* □

We note that by Theorem A.3.9 this is equivalent to proving that for each p and d there is an irreducible polynomial $m(x)$ in $\mathbb{F}_p[x]$ of degree d.

How do we actually find and calculate in finite fields? Theorem A.3.9 gives the answer. If we want a field F with p^d elements (usually written as $F = GF(p^d)$ or $F = \mathbb{F}_{p^d}$), then we first find an irreducible polynomial $m(x)$ of degree d in $\mathbb{F}_p[x]$ and then realize F as $\mathbb{F}_p[x] \pmod{m(x)}$.

We can check for irreducibility of a given polynomial in a way similar to the Sieve of Eratosthenes — if a polynomial of degree d is reducible, then it must be a multiple of an irreducible polynomial of degree at most $d/2$. For example $x^3 + x + 1 \in \mathbb{F}_2[x]$ is irreducible as it has no nonscalar factor of degree at most $3/2$, that is, it has no linear factors (as it has no roots in $\mathbb{F}_2$). Therefore even though Theorem A.3.10 is quite difficult to prove, it may not too hard to find an irreducible polynomial of a specific desired degree d in $\mathbb{F}_p[x]$. To do so, use the sieve to find all reducible polynomials of degree d, then all the remaining polynomials are irreducible. (There are only finitely many polynomials of a fixed degree in $\mathbb{F}_p[x]$.)

(A.3.11) PROBLEM. (*a*) *Find all irreducible polynomials of degree* 4 *or less in* $\mathbb{F}_2[x]$.
(*b*) *Find all monic irreducible polynomials of degree* 3 *or less in* $\mathbb{F}_3[x]$.
(*c*) *Find all monic irreducible polynomials of degree* 2 *or less in* $\mathbb{F}_4[x]$.
(*d*) *Find all monic irreducible polynomials of degree* 2 *or less in* $\mathbb{F}_5[x]$.

For notational elegance, we usually do not write F as $\mathbb{F}_p[x] \pmod{m(x)}$, but instead as the collection of polynomials of degree less than d in ρ, a root of the degree d irreducible $m(x)$. So, for example, rather than write the complex numbers as $\mathbb{R}[x] \pmod{x^2+1}$ we write them as the set of all $a + bi,\ a, b \in \mathbb{R}$, where i is a root of the irreducible polynomial $x^2 + 1$ of degree 2.

At the end of this section we give an example of a field with 32 elements, $\mathbb{F}_{32}$, written as polynomials of degree less than 5 in a root α of the primitive polynomial $x^5 + x^2 + 1 \in \mathbb{F}_2[x]$. Notice that as α is primitive, we may also write the nonzero elements of $\mathbb{F}_{32}$ as powers of α. This is helpful, because addition in $\mathbb{F}_{32}$ is easily done in terms of the polynomials of degree less than 5 in α, while multiplication is more easily done in terms of the powers of α.

(A.3.12) PROBLEM. (a) *Prove that the polynomial* $x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$ *is irreducible but not primitive.*

(b) *Let* β *be a root of the primitive polynomial* $x^4 + x^3 + 1 \in \mathbb{F}_2[x]$. *Write out a table of the elements of a field with* 16 *elements,* $\mathbb{F}_{16}$, *both as powers of* β *and as polynomials of degree less than* 4 *in* β.

The following simple result about finite fields is of great importance.

(A.3.13) LEMMA. *Let K be a field of characteristic p and J a subfield of K.*
(1) *If q is any power of p, then for any $a, b \in K$ we have $(a+b)^q = a^q + b^q$.*
(2) *If $|J| = q$ then $a^q = a$, for all $a \in J$, and J is the complete set of solutions to the equation $x^q = x$ in K.*

PROOF. (1) As $(c^p)^p = c^{p^2}, (c^{p^2})^p = c^{p^3}, \ldots$, we need only prove (1) for $q = p$. In that case it follows easily as each binomial coefficient $\binom{p}{i}$ is 0 modulo p, for $0 < i < p$.
(2) By Theorem A.3.8 $a^q = a$ for all $a \in J$. By Proposition A.2.10 $x^q - x$ has at most q roots in K, and these are exactly the members of J. □

Let D be a subfield of the finite field F, and assume that $D = \mathbb{F}_q$. As F can be viewed as a vector space over D, we must have $F = \mathbb{F}_{q^m}$, for some m. Define the *trace* from F to D of the element $\alpha \in F$ by trace

$$Tr_D(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{m-1}}.$$

If D is the prime subfield $\mathbb{F}_p$, we often drop the subscript and write Tr for $Tr_{\mathbb{F}_p}$.

(A.3.14) PROPOSITION. (1) *The trace is a map from F onto D.*
(2) *The trace is a D-linear; that is, for all $r_1, r_2 \in D$ and $\alpha_1, \alpha_2 \in F$, we have*

$$Tr_D(r_1\alpha_1 + r_2\alpha_2) = r_1 Tr_D(\alpha_1) + r_2 Tr_D(\alpha_2).$$

(3) *For a fixed $\beta \in F$, if $Tr_D(\alpha\beta) = 0$ for all α in a D-basis of F, then $\beta = 0$.*

PROOF. It is elementary to prove that the trace is a linear map into D as in (2) using Lemma A.3.13. It is not so clear that the map is actually onto D. The trace is given by a polynomial of degree q^{m-1}, so by Proposition A.2.10 there are at most q^{m-1} elements of F with trace 0. Since the trace is linear, the subset K of elements of F with trace 0 is a D-subspace of F, and the value of the trace map is constant on cosets $\alpha + K$ of K. Again by linearity, different cosets of K give different values. As $|F| = q^m$, there must be the largest possible number $q = |D|$ of values and cosets, and each coset must have the largest possible size, q^{m-1}. This gives (1).
By linearity, if $Tr_D(\alpha\beta) = 0$, for all α in a D-basis for F, then in fact $Tr_D(\alpha\beta) = 0$, for all $\alpha \in F$. But for $\beta \neq 0$, by (1) there are many choices of α with $Tr_D(\alpha\beta) \neq 0$, proving (3). □

(A.3.15) PROBLEM. *Let $T\colon F \to D$ be a D-linear map, that is,*

$$T(r_1\alpha_1 + r_2\alpha_2) = r_1 T(\alpha_1) + r_2 T(\alpha_2);$$

and define the map $B\colon F \times F \to D$ by $B(\alpha, \beta) = T(\alpha\beta)$.
(a) *Prove that B is a symmetric D-bilinear map; that is,*

$$B(\alpha, \beta) = B(\beta, \alpha) \text{ and}$$

$$B(r_1\alpha_1 + r_2\alpha_2, \beta) = r_1 B(\alpha_1, \beta) + r_2 B(\alpha_2, \beta), \text{ for all } r_1, r_2 \in D\,.$$

(*b*) *Prove that, conversely, every symmetric D-bilinear map B arises in this fashion from a D-linear map T. (*HINT: *Prove that the map T given by $T(\alpha) = B(\alpha, 1)$ is D-linear.)*

(*c*) *Prove, for a fixed nonzero $\beta \in F$, that $B(\alpha, \beta) = 0$ for all α in a D-basis of F if and only if T is the 0 map, that is, the map that takes each element of F to 0.*

trace dual basis

Let $\alpha_1, \ldots, \alpha_m$ be a basis for F over D. The second basis $\beta_1, \ldots, \beta_m$ is *trace dual basis* to the first if $Tr_D(\alpha_i\beta_j)$ $(= B(\alpha_i, \beta_j))$ is 1 when $i = j$ and 0 when $i \neq j$. In the next result we see that a trace dual basis always exists.

(A.3.16) PROPOSITION. *Let D be a subfield of the finite field F, and let $\alpha_1, \ldots, \alpha_m$ be a basis for F over D.*

We let A be the $m \times m$ matrix whose $\{i, j\}$-entry is $Tr_D(\alpha_i\alpha_j)$. For the $m \times s$ matrix B let the $\{j, k\}$-entry be $b_{j,k} \in F$. Finally let $\beta_k = \sum_{j=1}^{m} b_{j,k}\alpha_j$.

(1) *The $\{i, k\}$-entry of the matrix product AB is $Tr_D(\alpha_i\beta_k)$.*

(2) *The matrix A is invertible.*

(3) *For $B = A^{-1}$, the basis $\beta_1, \ldots, \beta_m$ is trace dual to $\alpha_1, \ldots, \alpha_m$.*

PROOF. Part (1) follows by an elementary matrix calculation.

If A is not invertible, then we can find a nonzero column vector B (with $s = 1$) such that $AB = 0$. This would correspond to a nonzero $\beta \in F$ with $Tr_D(\alpha_i\beta) = 0$, for all i. By Proposition A.3.14(3) this can not happen. This gives (2), and (3) is immediate from (1) and (2). □

(A.3.17) PROBLEM. *Reprove Proposition A.3.16 starting with an arbitrary nonzero D-linear map T.*

(A.3.18) PROBLEM. *Let the field $\mathbb{F}_8$ be written as polynomials of degree less than 3 over $\mathbb{F}_2$ in the primitive element α, a root of $x^3 + x + 1$, so that $\alpha^3 = \alpha + 1$. The trace $Tr = Tr_{\mathbb{F}_2}$ from $\mathbb{F}_8$ to $\mathbb{F}_2$ is then given by*

$$Tr(\beta) = \beta + \beta^2 + \beta^4$$

for all $\beta \in \mathbb{F}_8$. Set $e_1 = \alpha^3$, $e_2 = \alpha^5$, $e_3 = \alpha^6$, so that e_1, e_2, e_3 form a basis for $\mathbb{F}_8$ over $\mathbb{F}_2$.

(*a*) *Prove that the basis e_1, e_2, e_3 is trace self-dual: $Tr(e_ie_j)$ is 1 if $i = j$ and is 0 if $i \neq j$.*

(*b*) *For each $r \in \mathbb{F}_8$, let $\hat{r}$ be defined by $\hat{r} = (a, b, c)$, where $r = ae_1 + be_2 + ce_3$, for $a, b, c \in \mathbb{F}_2$. Prove that, for all $r, s \in \mathbb{F}_8$,*

$$\begin{aligned} Tr(rs) &= \hat{r} \cdot \hat{s} \quad \text{(dot product)} \\ &= af + bg + ch \end{aligned}$$

if $\hat{r} = (a, b, c)$ and $\hat{s} = (f, g, h)$.

(*c*) *Let $\mathbf{x}, \mathbf{y}$ be vectors in $\mathbb{F}_8^n$. Define the vectors $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ by*

$$\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n) \text{ for } \mathbf{x} = (x_1, x_2, \ldots, x_n)\,,$$

$$\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n) \text{ for } \mathbf{y} = (y_1, y_2, \ldots, y_n)\,.$$

Show that if $\mathbf{x} \cdot \mathbf{y} = 0$ in $\mathbb{F}_8$, then $\hat{\mathbf{x}} \cdot \hat{\mathbf{y}} = 0$ in $\mathbb{F}_2$.

Table. $\mathbb{F}_{32}$ where α is a root of the polynomial $x^5 + x^2 + 1$

Power	Polynomial of degree less than 5 in α	5−tuple
0	0	00000
1	1	00001
α^1	α^1	00010
α^2	α^2	00100
α^3	α^3	01000
α^4	α^4	10000
α^5	$\alpha^2 + 1$	00101
α^6	$\alpha^3 + \alpha^1$	01010
α^7	$\alpha^4 + \alpha^2$	10100
α^8	$\alpha^3 + \alpha^2 + 1$	01101
α^9	$\alpha^4 + \alpha^3 + \alpha^1$	11010
α^{10}	$\alpha^4 + 1$	10001
α^{11}	$\alpha^2 + \alpha^1 + 1$	00111
α^{12}	$\alpha^3 + \alpha^2 + \alpha^1$	01110
α^{13}	$\alpha^4 + \alpha^3 + \alpha^2$	11100
α^{14}	$\alpha^4 + \alpha^3 + \alpha^2 + 1$	11101
α^{15}	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	11111
α^{16}	$\alpha^4 + \alpha^3 + \alpha^1 + 1$	11011
α^{17}	$\alpha^4 + \alpha^1 + 1$	10011
α^{18}	$\alpha^1 + 1$	00011
α^{19}	$\alpha^2 + \alpha^1$	00110
α^{20}	$\alpha^3 + \alpha^2$	01100
α^{21}	$\alpha^4 + \alpha^3$	11000
α^{22}	$\alpha^4 + \alpha^2 + 1$	10101
α^{23}	$\alpha^3 + \alpha^2 + \alpha^1 + 1$	01111
α^{24}	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	11110
α^{25}	$\alpha^4 + \alpha^3 + 1$	11001
α^{26}	$\alpha^4 + \alpha^2 + \alpha^1 + 1$	10111
α^{27}	$\alpha^3 + \alpha^1 + 1$	01011
α^{28}	$\alpha^4 + \alpha^2 + \alpha^1$	10110
α^{29}	$\alpha^3 + 1$	01001
α^{30}	$\alpha^4 + \alpha^1$	10010
α^{31}	1	00001

A.3.3. Minimal Polynomials

Let D be any field and F an extension field of D (that is, D is a subfield of F). If α is any element of F, then as in Section A.3.2 we consider the collection of polynomials that have α as a root:

$$I = \{\, p(x) \in D[x] \,|\, p(\alpha) = 0 \}\,.$$

It is possible for I to contain only the zero polynomial, an example being given by $D = \mathbb{Q}$, $F = \mathbb{R}$, $\alpha = \pi$. We are interested here in the case where F is finite, and there the argument of Lemma A.1.3 and Section A.3.2 shows that I must contain nonzero polynomials.

minimal polynomial

Assuming that I contains nonzero polynomials, we denote by $m_{\alpha,D}(x)$ the *minimal polynomial* of α over D, that is, the greatest common divisor of I. When D is the prime subfield (here, $\mathbb{F}_p$ for some prime p) we have abbreviated this to $m_\alpha(x)$. A minimal polynomial must always be irreducible.

For a finite collection S of nonzero polynomials, the least common multiple, $\mathrm{lcm}(S)$, was introduced in Problem A.2.19. When all the members of S are monic irreducible, the lcm is easy to calculate — it is just the product of all distinct members of S (see Problem A.2.25).

(A.3.19) LEMMA. *Let $\alpha, \beta, \ldots, \omega$ be members of the extension field F of the field D. Then the set*

$$J = \{\, p(x) \in D[x] \mid p(\alpha) = p(\beta) = \cdots = p(\omega) = 0 \,\}$$

consists precisely of all multiples of

$$g(x) = \mathrm{lcm}(m_{\alpha,D}(x), m_{\beta,D}(x), \ldots, m_{\omega,D}(x)).$$

PROOF. By the definition of a minimal polynomial, for each element γ of $\alpha, \beta, \ldots, \omega$, the set J consists of multiples of $m_{\gamma,D}(x)$. Therefore by the definition of least common multiples (see Problem A.2.19) all members of J are multiples of $g(x)$. On the other hand, any multiple of $g(x)$ has each of $\alpha, \beta, \ldots, \omega$ as a root and so is in J. □

The remark before Lemma A.3.19 shows that, in the computation of $g(x)$ the only difficult part is the calculation of the minimal polynomials over D of members of F. In Theorem A.3.20 and Problem A.3.21 we describe an easy way to do this for finite D. At the end of the section an example of such a calculation using Theorem A.3.20 is presented.

(A.3.20) THEOREM. *Let F be a finite field of characteristic p, and let α be a member of F. Then for*

$$A = \{\alpha^{p^i} \mid i = 0, 1, 2, \ldots \}$$

we have

$$m_\alpha(x) = \prod_{a \in A} (x - a)\ .$$

PROOF. Let $m(x) = m_\alpha(x) = \sum_i m_i x^i$ with each m_i in $\mathbb{F}_p$. As $m(\alpha) = 0$, also $(m(\alpha))^p = 0$. That is,

$$\begin{aligned} 0 &= (\sum m_i\alpha^i)^p = \sum (m_i\alpha^i)^p && \text{by } A.3.13(1) \\ &= \sum m_i^p \alpha^{ip} = \sum m_i(\alpha^p)^i && \text{by } A.3.13(2) \\ &= m(\alpha^p). \end{aligned}$$

Thus from $m(\alpha) = 0$ we may conclude that $m(\alpha^p) = 0$ and then that $m((\alpha^p)^p) = m(\alpha^{p^2}) = 0$; indeed $m(a) = 0$, for all $a \in A$. By Lemma A.2.8 $x - a$ divides $m(x)$ for each $a \in A$, and so by repeated application of Lemma A.2.9 we know that $\prod_{a\in A}(x - a)$ is in any event a divisor of $m(x)$ in $F[x]$. To complete a proof that $m(x) = \prod_{a\in A}(x - a)$ it is enough to show that $\prod_{a\in A}(x - a)$ in fact has all its coefficients in $\mathbb{F}_p$, for then $m(x)$ and $\prod_{a\in A}(x - a)$ will be two monic polynomials of $\mathbb{F}_p[x]$ that divide each other and so must be equal.

Let $A = \{a_1, a_2, \ldots, a_d\}$. Then in $\prod_{a\in A}(x - a)$ the coefficient of x^k is

$$\sum_{\{i_1,i_2,\ldots,i_{d-k}\}} a_{i_1} a_{i_2} \cdots a_{i_{d-k}} ,$$

where the summation runs over all $d-k$ subsets of $\{1, 2, \ldots, d\}$. By design, for each a_i in A, a_i^p is also a member of A. Therefore for each term $a_{i_1} a_{i_2} \cdots a_{i_{d-k}}$ of the above summation, the power $(a_{i_1} a_{i_2} \cdots a_{i_{d-k}})^p = a_{i_1}^p a_{i_2}^p \cdots a_{i_{d-k}}^p$ is also one of the terms of the summation. Hence using Lemma A.3.13(1) again we have

$$(\sum a_{i_1} a_{i_2} \cdots a_{i_{d-k}})^p = \sum a_{i_1}^p a_{i_2}^p \cdots a_{i_{d-k}}^p = \sum a_{i_1} a_{i_2} \cdots a_{i_{d-k}} .$$

That is, the coefficient of x^k in $\prod_{a\in A}(x - a)$ is equal to its own p^{th} power. By Lemma A.3.13(2) this coefficient is a member of the prime subfield $\mathbb{F}_p$, as required. □

Essentially the same proof with q in place of p gives the more general result (which we leave as an exercise) with $D = \mathbb{F}_q$ in place of $\mathbb{F}_p$:

(A.3.21) PROBLEM. *Let F be a finite field of characteristic p, D a subfield of F containing exactly q elements, and α be a member of F. Then for*

$$A = \{\alpha^{q^i} \mid i = 0, 1, 2, \ldots\}$$

we have

$$m_{\alpha,D}(x) = \prod_{a\in A}(x - a).$$

REMARK. At first sight, the final equations in the statement of Theorem A.3.20 and Problem A.3.21 seem to go against our claim that minimal polynomials must be irreducible. Here $m_{\alpha,D}(x)$ is a minimal polynomial, but $\prod_{a\in A}(x-a)$

appears to be a nontrivial factorization. The point is that $m_{\alpha,D}(x)$ is an irreducible polynomial in the polynomial ring $D[x]$; it has no factorizations into polynomials of $D[x]$ of smaller degree. The factorization $\prod_{a\in A}(x-a)$ involves factors $x-a$ that are polynomials of $F[x]$ but not of $D[x]$ (as long as $a \notin D$). For example, as a polynomial of $\mathbb{R}[x]$, x^2+1 is irreducible; but as a polynomial of $\mathbb{C}[x]$ it factors as $x^2+1=(x+i)(x-i)$. Indeed $m_{i,\mathbb{R}}(x)=x^2+1$.

Below we give an example which details the calculation using Theorem A.3.20 of the minimal polynomial of α^5 over $\mathbb{F}_2$, $m_{\alpha^5,\mathbb{F}_2}(x)$, where α is a root of the primitive polynomial $x^5+x^2+1 \in \mathbb{F}_2[x]$. (See the table at the end of Section A.3.2.)

(A.3.22) PROBLEM. *Let β be a root of the polynomial $x^4+x^3+1 \in \mathbb{F}_2[x]$. Calculate the minimal polynomial of β^3.*

Calculation of a minimal polynomial

Let α be a primitive element in $\mathbb{F}_{32}$ with minimal polynomial $m_\alpha(x)=m_{\alpha,\mathbb{F}_2}(x)=x^5+x^2+1$. We wish to calculate the minimal polynomial of α^5.

$$
\begin{aligned}
& m_{\alpha^5,\mathbb{F}_2}(x) \\
&\quad = (x-\alpha^5)(x-\alpha^{10})(x-\alpha^{20})(x-\alpha^9)(x-\alpha^{18}) \\
&\quad = x^5-(\alpha^5+\alpha^{10}+\alpha^{20}+\alpha^9+\alpha^{18})x^4 \\
&\qquad +(\alpha^{15}+\alpha^{25}+\alpha^{14}+\alpha^{23}+\alpha^{30}+\alpha^{19}+\alpha^{28}+\alpha^{29}+\alpha^{38}+\alpha^{27})x^3 \\
&\qquad -(\alpha^{47}+\alpha^{37}+\alpha^{48}+\alpha^{39}+\alpha^{32}+\alpha^{43}+\alpha^{34}+\alpha^{33}+\alpha^{24}+\alpha^{35})x^2 \\
&\qquad +(\alpha^{57}+\alpha^{52}+\alpha^{42}+\alpha^{53}+\alpha^{44})x-\alpha^{62} \\
&\quad = x^5+1x^4+0x^3+1x^2+1x+1 \\
&\quad = x^5+x^4+x^2+x+1\ .
\end{aligned}
$$

Where, for instance, the coefficient of x is given by:

$$
\begin{aligned}
& \alpha^{57}+\alpha^{52}+\alpha^{42}+\alpha^{53}+\alpha^{44} \\
&\quad = \alpha^{26}+\alpha^{21}+\alpha^{11}+\alpha^{22}+\alpha^{13} \\
&\quad = (\alpha^4+\alpha^2+\alpha+1)+(\alpha^4+\alpha^3)+(\alpha^2+\alpha+1) \\
&\qquad +(\alpha^4+\alpha^2+1)+(\alpha^4+\alpha^3+\alpha^2) \\
&\quad = 1\ .
\end{aligned}
$$