Algebraic Codes for Data Transmission

Richard E. Blahut

Henry Magnuski Professor in Electrical and Computer Engineering,
University of Illinois at Urbana – Champaign

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2003

Printed in the United Kingdom at the University Press, Cambridge

Typefaces Times 10.5/14 pt and Helvetica Neue *System* LaTeX 2$_\varepsilon$ [TB]

A catalogue record for this book is available from the British Library

ISBN 0 521 55374 1 hardback

Contents

Preface *page* xi

1 Introduction 1

1.1 The discrete communication channel 2
1.2 The history of data-transmission codes 4
1.3 Applications 6
1.4 Elementary concepts 7
1.5 Elementary codes 14
Problems 17

2 Introduction to Algebra 20

2.1 Fields of characteristic two 20
2.2 Groups 23
2.3 Rings 28
2.4 Fields 30
2.5 Vector spaces 32
2.6 Linear algebra 37
Problems 45
Notes 48

3 Linear Block Codes 49

3.1 Structure of linear block codes 49
3.2 Matrix description of linear block codes 50
3.3 Hamming codes 54
3.4 The standard array 56

3.5 Hamming spheres and perfect codes 59
3.6 Simple modifications to a linear code 62
Problems 63
Notes 66

4 The Arithmetic of Galois Fields 67

4.1 The integer ring 67
4.2 Finite fields based on the integer ring 70
4.3 Polynomial rings 72
4.4 Finite fields based on polynomial rings 79
4.5 Primitive elements 83
4.6 The structure of finite fields 86
Problems 92
Notes 95

5 Cyclic Codes 96

5.1 Viewing a code from an extension field 96
5.2 Polynomial description of cyclic codes 99
5.3 Minimal polynomials and conjugates 104
5.4 Matrix description of cyclic codes 111
5.5 Hamming codes as cyclic codes 113
5.6 Cyclic codes for correcting double errors 116
5.7 Quasi-cyclic codes and shortened cyclic codes 118
5.8 The Golay code as a cyclic code 119
5.9 Cyclic codes for correcting burst errors 123
5.10 The Fire codes as cyclic codes 125
5.11 Cyclic codes for error detection 127
Problems 128
Notes 130

6 Codes Based on the Fourier Transform 131

6.1 The Fourier transform 131
6.2 Reed–Solomon codes 138

6.3 Conjugacy constraints and idempotents 143
6.4 Spectral description of cyclic codes 148
6.5 BCH codes 152
6.6 The Peterson–Gorenstein–Zierler decoder 159
6.7 The Reed–Muller codes as cyclic codes 166
6.8 Extended Reed–Solomon codes 169
6.9 Extended BCH codes 172
Problems 175
Notes 177

7 Algorithms Based on the Fourier Transform 179

7.1 Spectral estimation in a finite field 179
7.2 Synthesis of linear recursions 183
7.3 Decoding of binary BCH codes 191
7.4 Decoding of nonbinary BCH codes 193
7.5 Decoding with erasures and errors 201
7.6 Decoding in the time domain 206
7.7 Decoding within the BCH bound 210
7.8 Decoding beyond the BCH bound 213
7.9 Decoding of extended Reed–Solomon codes 216
7.10 Decoding with the euclidean algorithm 217
Problems 223
Notes 226

8 Implementation 228

8.1 Logic circuits for finite-field arithmetic 228
8.2 Shift-register encoders and decoders 235
8.3 The Meggitt decoder 237
8.4 Error trapping 244
8.5 Modified error trapping 250
8.6 Architecture of Reed–Solomon decoders 254
8.7 Multipliers and inverters 258
8.8 Bit-serial multipliers 262
Problems 267
Notes 269

9 Convolutional Codes 270

9.1 Codes without a block structure 270
9.2 Trellis description of convolutional codes 273
9.3 Polynomial description of convolutional codes 278
9.4 Check matrices and inverse matrices 282
9.5 Error correction and distance notions 287
9.6 Matrix description of convolutional codes 289
9.7 The Wyner–Ash codes as convolutional codes 291
9.8 Syndrome decoding algorithms 294
9.9 Convolutional codes for correcting error bursts 298
9.10 Algebraic structure of convolutional codes 303
Problems 309
Notes 311

10 Beyond BCH Codes 313

10.1 Product codes and interleaved codes 314
10.2 Bicyclic codes 318
10.3 Concatenated codes 321
10.4 Cross-interleaved codes 323
10.5 Turbo codes 326
10.6 Justesen codes 329
Problems 332
Notes 334

11 Codes and Algorithms Based on Graphs 335

11.1 Distance, probability, and likelihood 336
11.2 The Viterbi algorithm 340
11.3 Sequential algorithms to search a trellis 343
11.4 Trellis description of linear block codes 350
11.5 Gallager codes 354
11.6 Tanner graphs and factor graphs 355
11.7 Posterior probabilities 357
11.8 The two-way algorithm 359
11.9 Iterative decoding of turbo codes 362

11.10 Tail-biting representations of block codes 364
11.11 The Golay code as a tail-biting code 368
Problems 372
Notes 374

12 Performance of Error-Control Codes 375

12.1 Weight distributions of block codes 375
12.2 Performance of block codes 383
12.3 Bounds on minimum distance of block codes 386
12.4 Binary expansions of Reed–Solomon codes 394
12.5 Symbol error rates on a gaussian-noise channel 399
12.6 Sequence error rates on a gaussian-noise channel 403
12.7 Coding gain 406
12.8 Capacity of a gaussian-noise channel 411
Problems 414
Notes 416

13 Codes and Algorithms for Majority Decoding 418

13.1 Reed–Muller codes 418
13.2 Decoding by majority vote 426
13.3 Circuits for majority decoding 430
13.4 Affine permutations for cyclic codes 433
13.5 Cyclic codes based on permutations 437
13.6 Convolutional codes for majority decoding 441
13.7 Generalized Reed–Muller codes 442
13.8 Euclidean-geometry codes 447
13.9 Projective-geometry codes 456
Problems 460
Notes 461

Bibliography 463
Index 473

1 Introduction

A profusion and variety of communication systems, which carry massive amounts of digital data between terminals and data users of many kinds, exist today. Alongside these communication systems are many different magnetic tape storage systems, and magnetic and optical disk storage systems. The received signal in any communication or recording system is always contaminated by thermal noise and, in practice, may also be contaminated by various kinds of defects, nongaussian noise, burst noise, interference, fading, dispersion, cross talk, and packet loss. The communication system or storage system must transmit its data with very high reliability in the presence of these channel impairments. Bit error rates as small as one bit error in 10^{12} bits (or even smaller) are routinely specified.

Primitive communication and storage systems may seek to keep bit error rates small by the simple expedient of transmitting high signal power or by repeating the message. These simplistic techniques may be adequate if the required bit error rate is not too stringent, or if the data rate is low, and if errors are caused by noise rather than by defects or interference. Such systems, however, buy performance with the least expendable resources: Power and bandwidth.

In contrast, modern communication and storage systems obtain high performance via the use of elaborate message structures with complex cross-checks built into the waveform. The advantage of these modern communication waveforms is that high data rates can be reliably transmitted while keeping the transmitted power and spectral bandwidth small. This advantage is offset by the need for sophisticated computations in the receiver (and in the transmitter) to recover the message. Such computations, however, are now regarded as affordable by using modern electronic technology. For example, current telephone-line data modems use microprocessors in the demodulator with well over 500 machine cycles of computation per received data bit. Clearly, with this amount of computation in the modem, the waveforms may have a very sophisticated structure, allowing each individual bit to be deeply buried in the waveform. In some systems it may be impossible to specify where a particular user bit resides in the channel waveform; the entire message is modulated into the channel waveform as a package, and an individual bit appears in a diffuse but recoverable way.

The data-transmission codes described in this book are codes used for the prevention of error. The phrase "prevention of error" has a positive tone that conveys the true role such codes have in modern systems. The more neutral term, "error-control code," is also suitable. The older and widespread term, "error-correcting code," is used as well, but suffers from the fact that it has a negative connotation. It implies that the code is used only to correct an unforeseen deficiency in the communication system whereas, in modern practice, the code is an integral part of any high-performance communication or storage system. Furthermore, in many applications, the code is so tightly integrated with the demodulation that the point within the system where the errors occur and are corrected is really not visible to any external observer. It is a better description to say that the errors are prevented because the preliminary estimates of the data bits within the receiver are accompanied by extra information that cross-checks these data bits. In this sense, the errors never really happen because they are eliminated when the preliminary estimate of the datastream is replaced by the final estimate of the datastream that is given to the user.

1.1 The discrete communication channel

A communication system connects a data source to a data user through a channel. Microwave links, coaxial cables, telephone circuits, and even magnetic and optical disks are examples of channels. A discrete communication channel may transmit binary symbols, or symbols in an alphabet of size 2^m, or even symbols in an alphabet of size q where q is not a power of 2. Indeed, digital communication theory teaches that discrete channels using a larger symbol alphabet are usually more energy efficient than channels that use a binary alphabet.

The designer of the communication system develops devices that prepare the codestream for the input to the discrete channel and process the output of the discrete channel to recover the user's datastream. Although user data may originate as a sequence of bits, within the communication system it is often treated as a sequence of symbols. A symbol may consist of eight bits; then it is called a *byte*. In other cases, a communication system may be designed around a symbol of r bits for some value of r other than eight; the symbol then is called an r-bit symbol. The choice of symbol structure within the communication system is transparent to the user because the datastream is reformatted at the input and output of the communication system.

A *datastream* is a sequence of data symbols, which could be bits, bytes, or other symbols at the input of an encoder. A *codestream* is a sequence of channel symbols, which could be bits, bytes, or other symbols at the output of an encoder. The user perceives that the datastream is being sent through the channel, but what is actually sent is the codestream.

The encoder maps the datastream into the codestream. Codes are of two types: block codes and tree codes. The distinction between them is based on the way that data

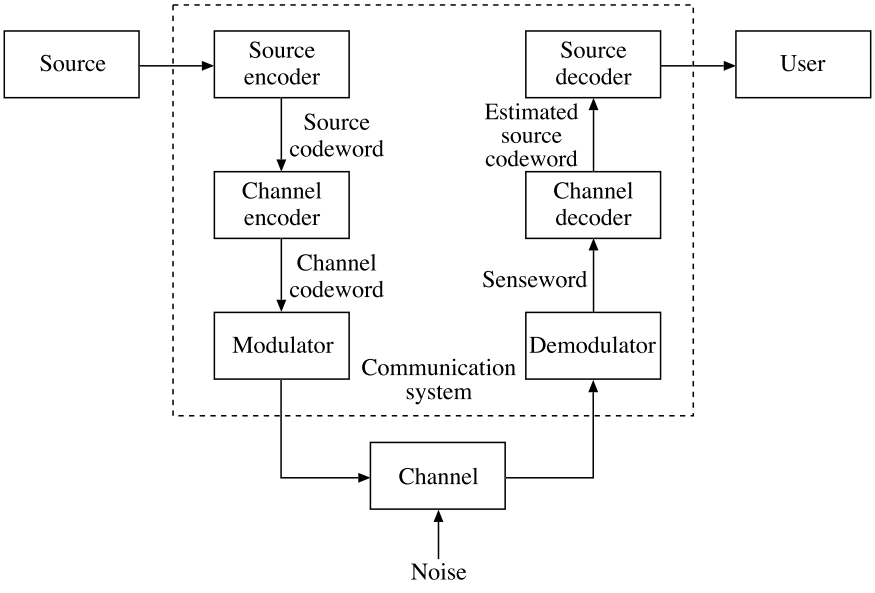


Figure 1.1. Block diagram of a digital communication system

memory is used in the encoder. For constructing the codestream, additional structure is defined on the datastream by segmenting it into pieces called *datawords* or *dataframes*. Likewise, the codestream is segmented into pieces called *codewords* or *codeframes*. The codewords or codeframes are serially concatenated to form the codestream.

It is traditional to partition the major functions of the digital communication system as in the block diagram of Figure 1.1. Data, which enters the communication system from the data source, is first processed by a source encoder designed to represent the source data more compactly. This interim representation is a sequence of symbols called the *source codestream*. The source codestream becomes the input datastream to the channel encoder, which transforms the sequence of symbols into another sequence called the *channel codestream*. The channel codestream is a new, longer sequence that has more redundancy than the source codestream. Each symbol in the channel codestream might be represented by a bit, or perhaps by a group of bits. Next, the modulator converts each symbol of the channel codestream into a corresponding symbol from a finite set of symbols known as the channel alphabet. This sequence of analog symbols from the channel alphabet is transmitted through the channel.

Because the channel is subject to various types of noise, distortion, and interference, the channel output differs from the channel input. The demodulator may convert the received channel output signal into a sequence of the symbols of the channel codestream. Then each demodulated symbol is a best estimate of that code symbol, though the demodulator may make some errors because of channel noise. The demodulated sequence of symbols is called the *senseword* or the *received word*. Because of errors, the symbols of the senseword do not always match those of the channel codestream. The channel decoder uses the redundancy in the channel codestream to correct the errors

in the received word and then produces an estimate of the user datastream. If all errors are corrected, the estimated user datastream matches the original user datastream. The source decoder performs the inverse operation of the source encoder and delivers its output datastream to the user.

Alternatively, some functions of the demodulator may be moved into the channel decoder in order to improve performance. Then the demodulator need not make hard decisions on individual code symbols but may give the channel decoder something closer to the raw channel data.

This book deals only with the design of the channel encoder and decoder, a subject known as the subject of *error-control codes*, or *data-transmission codes*, or perhaps, *error-prevention codes*. The emphasis is on the algebraic aspects of the subject; the interplay between algebraic codes and modulation is treated only lightly. The data compression or data compaction functions performed by the source encoder and source decoder are not discussed within this book, nor are the modulator and the demodulator. The channel encoder and the channel decoder will be referred to herein simply as the encoder and the decoder, respectively.

1.2 The history of data-transmission codes

The history of data-transmission codes began in 1948 with the publication of a famous paper by Claude Shannon. Shannon showed that associated with any communication channel or storage channel is a number C (measured in bits per second), called the *capacity* of the channel, which has the following significance. Whenever the information transmission rate R (in bits per second) required of a communication or storage system is less than C then, by using a data-transmission code, it is possible to design a communication system for the channel whose probability of output error is as small as desired. In fact, an important conclusion from Shannon's theory of information is that it is wasteful to make the raw error rate from an uncoded modulator–demodulator too good; it is cheaper and ultimately more effective to use a powerful data-transmission code.

Shannon, however, did not tell us how to find suitable codes; his contribution was to prove that they exist and to define their role. Throughout the 1950s, much effort was devoted to finding explicit constructions for classes of codes that would produce the promised arbitrarily small probability of error, but progress was meager. In the 1960s, for the most part, there was less obsession with this ambitious goal; rather, coding research began to settle down to a prolonged attack along two main avenues.

The first avenue has a strong algebraic flavor and is concerned primarily with block codes. The first block codes were introduced in 1950 when Hamming described a class of single-error-correcting block codes. Shortly thereafter Muller (1954) described a class of multiple-error-correcting codes and Reed (1954) gave a decoding algorithm for them. The Hamming codes and the Reed–Muller codes were disappointingly weak

compared with the far stronger codes promised by Shannon. Despite diligent research, no better class of codes was found until the end of the decade. During this period, codes of short blocklength were found, but without any general theory. The major advances came when Bose and Ray-Chaudhuri (1960) and Hocquenghem (1959) found a large class of multiple-error-correcting codes (the BCH codes), and Reed and Solomon (1960) and, independently, Arimoto (1961) found a related class of codes for nonbinary channels. Although these remain among the most important classes of codes, the theory of the subject since that time has been greatly strengthened, and new codes continue to be discovered.

The discovery of BCH codes led to a search for practical methods of designing the hardware or software to implement the encoder and decoder. The first good algorithm was found by Peterson (1960). Later, a powerful algorithm for decoding was discovered by Berlekamp (1968) and Massey (1969), and its implementation became practical as new digital technology became available. Now many varieties of algorithms are available to fit different codes and different applications.

The second avenue of coding research has a more probabilistic flavor. Early research was concerned with estimating the error probability for the best family of block codes despite the fact that the best codes were not known. Associated with these studies were attempts to understand encoding and decoding from a probabilistic point of view, and these attempts led to the notion of sequential decoding. Sequential decoding required the introduction of a class of nonblock codes of indefinite length, which can be represented by a tree and can be decoded by algorithms for searching the tree. The most useful tree codes are highly structured codes called *convolutional codes*. These codes can be generated by a linear shift-register circuit that performs a convolution operation on the data sequence. Convolutional codes were successfully decoded by sequential decoding algorithms in the late 1950s. It is intriguing that the Viterbi algorithm, a much simpler algorithm for decoding them, was not developed until 1967. The Viterbi algorithm gained widespread popularity for convolutional codes of modest complexity, but it is impractical for stronger convolutional codes.

During the 1970s, these two avenues of research began to draw together in some ways and to diverge further in others. Development of the algebraic theory of convolutional codes was begun by Massey and Forney, who brought new insights to the subject of convolutional codes. In the theory of block codes, schemes were proposed to construct good codes of long blocklength. Concatenated codes were introduced by Forney (1966), and Justesen used the idea of a concatenated code to devise a completely constructive class of long block codes with good performance. Meanwhile, Goppa (1970) defined a class of codes that is sure to contain good codes, though without saying how to identify the good ones.

The 1980s saw encoders and decoders appear frequently in newly designed digital communication systems and digital storage systems. A visible example is the compact disk, which uses a simple Reed–Solomon code for correcting double byte errors. Reed–Solomon codes also appear frequently in many magnetic tape drives and network

modems, and now in digital video disks. In other applications, such as telephone-line modems, the role of algebraic codes has been displaced by euclidean-space codes, such as the trellis-coded modulation of Ungerboeck (1982). The success of these methods led to further work on the design of nonalgebraic codes based on euclidean distance. The decade closed with widespread applications of data-transmission codes. Meanwhile, mathematicians took the search for good codes based on the Hamming distance into the subject of algebraic geometry and there started a new wave of theoretical progress that continues to grow.

The 1990s saw a further blurring of the walls between coding, signal processing, and digital communications. The development of the notion of turbo decoding and the accompanying codes of Berrou (1993) can be seen as the central event of this period. This work did as much for communications over the wideband channel as Ungerboeck's work did the previous decade for communications over the bandlimited channel. Practical iterative algorithms, such as the "two-way algorithm," for soft-decision decoding of large binary codes are now available to achieve the performance promised by Shannon. The Ungerboeck codes and the Berrou codes, together with their euclidean-space decoding algorithms, have created a body of techniques, still in rapid development, that lie midway between the subjects of modulation theory and of data transmission codes. Further advances toward the codes promised by Shannon are awaited.

This decade also saw the development of algorithms for hard-decision decoding of large nonbinary block codes defined on algebraic curves. Decoders for the codes known as hermitian codes are now available and these codes may soon appear in commercial products. At the same time, the roots of the subject are growing even deeper into the rich soil of mathematics.

1.3 Applications

Because the development of data-transmission codes was motivated primarily by problems in communications, much of the terminology of the subject has been drawn from the subject of communication theory. These codes, however, have many other applications. Codes are used to protect data in computer memories and on digital tapes and disks, and to protect against circuit malfunction or noise in digital logic circuits.

Applications to communication problems are diversified. Binary messages are commonly transmitted between computer terminals, in communication networks, between aircraft, and from spacecraft. Codes can be used to achieve reliable communication even when the received signal power is close to the thermal noise power. And, as the electromagnetic spectrum becomes ever more crowded with man-made signals, data-transmission codes will become even more important because they permit communication links to function reliably in the presence of interference. In military applications, it often is essential to employ a data-transmission code to protect against intentional enemy interference.

Many communication systems have limitations on transmitted power. For example, power may be very expensive in communication relay satellites. Data-transmission codes provide an excellent tool with which to reduce power needs because, with the aid of the code, the messages received weakly at their destinations can be recovered correctly.

Transmissions within computer systems usually are intolerant of even very low error rates because a single error can destroy the validity of a computer program. Error-control coding is important in these applications. Bits can be packed more tightly into some kinds of computer memories (magnetic or optical disks, for example) by using a data-transmission code.

Another kind of communication system structure is a multiaccess system, in which each of a number of users is preassigned an access slot for the channel. This access may be a time slot or frequency slot, consisting of a time interval or frequency interval during which transmission is permitted, or it may be a predetermined coded sequence representing a particular symbol that the user is permitted to transmit. A long binary message may be divided into packets with one packet transmitted within an assigned access slot. Occasionally packets become lost because of collisions, synchronization failure, or routing problems. A suitable data-transmission code protects against these losses because missing packets can be deduced from known packets.

Communication is also important within a large system. In complex digital systems, a large data flow may exist between subsystems. Digital autopilots, digital process-control systems, digital switching systems, and digital radar signal processing all are systems that involve large amounts of digital data which must be shared by multiple interconnected subsystems. This data transfer might be either by dedicated lines or by a more sophisticated, time-shared data-bus system. In either case, error-control techniques are important to ensure proper performance.

Eventually, data-transmission codes and the circuits for encoding and decoding will reach the point where they can handle massive amounts of data. One may anticipate that such techniques will play a central role in all communication systems of the future. Phonograph records, tapes, and television waveforms of the near future will employ digital messages protected by error-control codes. Scratches in a record, or interference in a received signal, will be completely suppressed by the coding as long as the errors are less serious than the capability designed into the error-control code. (Even as these words were written for the first edition in 1981, the as yet unannounced compact disk was nearing the end of its development.)

1.4 Elementary concepts

The subject of data-transmission codes is both simple and difficult at the same time. It is simple in the sense that the fundamental problem is easily explained to any technically

trained person. It is difficult in the sense that the development of a solution – and only a partial solution at that – occupies the length of this book. The development of the standard block codes requires a digression into topics of modern algebra before it can be studied.

Suppose that all data of interest can be represented as binary (coded) data, that is, as a sequence of zeros and ones. This binary data is to be transmitted through a binary channel that causes occasional errors. The purpose of a code is to add extra check symbols to the data symbols so that errors may be found and corrected at the receiver. That is, a sequence of data symbols is represented by some longer sequence of symbols with enough redundancy to protect the data.

A binary code of size M and *blocklength* n is a set of M binary words of length n called *codewords*. Usually, $M = 2^k$ for an integer k, and the code is referred to as an (n, k) binary code.

For example, we can make up the following code

$$\mathcal{C} = \left\{ \begin{matrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{matrix} \right\}.$$

This is a very poor (and very small) code with $M = 4$ and $n = 5$, but it satisfies the requirements of the definition, so it is a code. We can use this code to represent two-bit binary numbers by using the following (arbitrary) correspondence:

$$\begin{matrix} 0 & 0 & \leftrightarrow & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & \leftrightarrow & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & \leftrightarrow & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & \leftrightarrow & 1 & 1 & 1 & 1 & 1. \end{matrix}$$

If one of the four five-bit codewords is received, we may then suppose that the corresponding two data bits are the original two data bits. If an error is made, we receive a different five-bit senseword. We then attempt to find the most likely transmitted codeword to obtain our estimate of the original two data bits.

For example, if we receive the senseword $(0, 1, 1, 0, 0)$, then we may presume that $(0, 1, 1, 1, 0)$ was the transmitted codeword, and hence 10 is the two-bit dataword. If we receive the "soft" senseword consisting of the real numbers $(0.1, 1.1, 0.9, 0.4, 0.2)$ then we may presume that $(0, 1, 1, 1, 0)$ was the transmitted codeword because it is closest in euclidean distance, and hence 10 is the two-bit dataword. The decoding of soft sensewords is treated in Chapter 11.

The code of the example is not a good code because it is not able to correct many patterns of errors. We want to design a code so that every codeword is as different as possible from every other codeword, and we want to do this especially when the blocklength is long.

The first purpose of this book is to find good codes. Although, superficially, this may seem like a simple task, it is, in fact, exceedingly difficult, and many good codes are as yet undiscovered.

To the inexperienced, it may seem that it should suffice to define the requirements of a good code and then let a computer search through the set of all possible codes. But how many binary codes are there for a given (n, k)? Each codeword is a sequence of n binary symbols, and there are 2^k such codewords in an (n, k) binary code. Therefore a code is described by $n \cdot 2^k$ binary symbols. Altogether there are $2^{n \cdot 2^k}$ ways of picking these binary symbols. Hence the number of different (n, k) codes is $2^{n \cdot 2^k}$. Of course, a great many of these codes are of little value (as when two codewords are identical), but either the computer search must include these codes or some theory must be developed for excluding them.

For example, take $(n, k) = (40, 20)$, which is a very modest code by today's standards. The number of such codes is much larger than $10^{10,000,000}$ – an inconceivably large number. Hence undisciplined search procedures are worthless.

In general, we define block codes over an arbitrary finite alphabet, say the alphabet with q symbols $\{0, 1, 2, \ldots, q-1\}$. At first sight, it might seem to be an unnecessary generalization to introduce alphabets other than the binary alphabet. For reasons such as energy efficiency, however, many channels today are nonbinary, and codes for these channels must be nonbinary. In fact, data-transmission codes for nonbinary channels are often quite good, and this can reinforce the reasons for using a nonbinary channel. It is a trivial matter to represent binary source data in terms of a q-ary alphabet, especially if q is a power of 2, as usually it is in practice.

Definition 1.4.1. *A block code of size M over an alphabet with q symbols is a set of M q-ary sequences of length n called* codewords.

If $q = 2$, the symbols are called bits. Usually, $M = q^k$ for some integer k, and we shall be interested only in this case, calling the code an (n, k) code. Each sequence of k q-ary data symbols can be associated with a sequence of n q-ary symbols comprising a codeword.

There are two basic classes of codes: *block codes* and *trellis codes*. These are illustrated in Figure 1.2. A block code represents a block of k data symbols by an n-symbol codeword. The rate R of a block code[1] is defined as $R = k/n$. Initially, we shall restrict our attention to block codes.

A trellis code is more complicated. It takes a nonending sequence of data symbols arranged in k-symbol segments called *dataframes*, and puts out a continuous sequence of code symbols arranged in n-symbol segments called *codeframes*. The distinction with

[1] This rate is dimensionless, or perhaps measured in units of bits/bit or symbols/symbol. It should be distinguished from another use of the term *rate* measured in bits/second through a channel. Yet another definition, $R = (k/n)\log_e q$, which has the units of nats/symbol, with a nat equaling $\log_2 e$ bits, is in use. The definition $R = (k/n)\log_2 q$, which has the units of bits/symbol, is also popular.

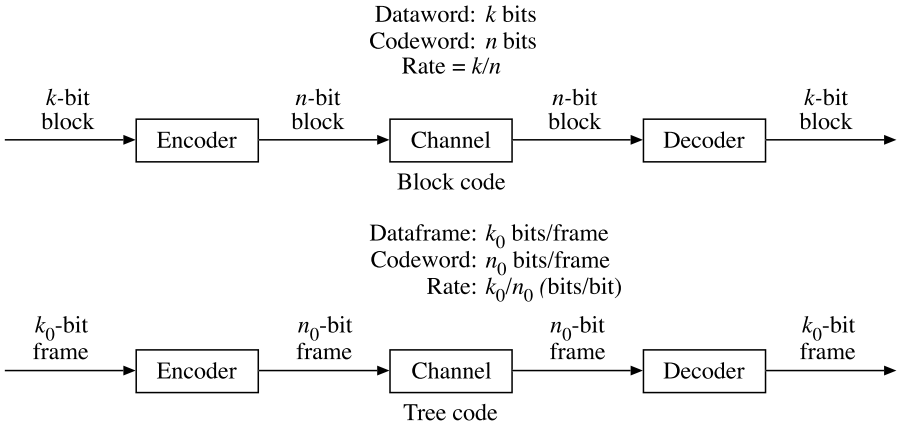


Figure 1.2. Basic classes of codes

block codes is that in a trellis code, a k-symbol dataframe can affect all succeeding codeword frames, whereas in a block code, a k-symbol datablock determines only the next n-symbol codeblock, but no others. We shall defer the study of trellis codes, specifically convolutional codes, until Chapter 9.

Whenever a message consists of a large number of bits, it is better, in principle, to use a single block code of large blocklength than to use a succession of codewords from a shorter block code. The nature of statistical fluctuations is such that a random pattern of errors usually exhibits some clustering of errors. Some segments of the random pattern contain more than the average number of errors, and some segments contain less. Long codewords are considerably less sensitive to random errors than are short codewords of the same rate, because a segment with many errors can be offset by a segment with few errors, but of course, the encoder and decoder may be more complex.

As an example, suppose that 1000 data bits are transmitted with a (fictitious) 2000-bit binary codeword that can correct 100 bit errors. Compare this with a scheme for transmitting 100 data bits at a time with a 200-bit binary codeword that can correct 10 bit errors per block. Ten such blocks are needed to transmit 1000 bits. This latter scheme can also correct a total of 100 errors, but only if they are properly distributed – ten errors to a 200-bit block. The first scheme can correct 100 errors no matter how they are distributed within the 2000-bit codeword. It is far more powerful.

This heuristic argument can be given a sound theoretical footing, but that is not our purpose here. We only wish to make plausible the fact that good codes are of long blocklength, and that very good codes are of very long blocklength. Such codes can be very hard to find and, when found, may require complex devices to implement the encoding and decoding operations.

Given two sequences of the same length of symbols from some fixed symbol alphabet, perhaps the binary alphabet $\{0, 1\}$, we shall want to measure how different those two sequences are from each other. The most suggestive way to measure the difference

between the two sequences is to count the number of places in which they differ. This is called the *Hamming distance* between the sequences.

Definition 1.4.2. *The Hamming distance $d(\boldsymbol{x}, \boldsymbol{y})$ between two q-ary sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ of length n is the number of places in which $\boldsymbol{x}$ and $\boldsymbol{y}$ differ.*

For example, take $\boldsymbol{x} = 10101$, $\boldsymbol{y} = 01100$, then $d(10101, 01100) = 3$. For another example, take $\boldsymbol{x} = 30102$, $\boldsymbol{y} = 21103$, then $d(30102, 21103) = 3$.

The reason for choosing the term "distance" is to appeal to geometric intuition when constructing codes. It is obvious that the Hamming distance is nonnegative and symmetric. It is easy to verify that the Hamming distance also satisfies the triangle inequality $d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{y}, \boldsymbol{z})$. This means that geometric reasoning and intuition based on these properties are valid.

Definition 1.4.3. *Let $\mathcal{C} = \{\boldsymbol{c}_\ell \mid \ell = 0, \ldots, M-1\}$ be a code. Then the minimum Hamming distance $d_{\min}$ (or d) of $\mathcal{C}$ is the Hamming distance between the pair of codewords with smallest Hamming distance. That is,*

$$d_{\min} = \min_{\substack{\boldsymbol{c}_i, \boldsymbol{c}_j \in \mathcal{C} \\ i \neq j}} d(\boldsymbol{c}_i, \boldsymbol{c}_j).$$

Block codes are judged by three parameters: the blocklength n, the datalength k, and the minimum distance $d_{\min}$. An (n, k) block code with minimum distance $d_{\min}$ is also described as an $(n, k, d_{\min})$ block code.

In the block code $\mathcal{C}$, given in the first example of this section,

$$d(10101, 10010) = 3$$
$$d(10101, 01110) = 4$$
$$d(10101, 11111) = 2$$
$$d(10010, 01110) = 3$$
$$d(10010, 11111) = 3$$
$$d(01110, 11111) = 2.$$

Hence $d_{\min} = 2$ for this code.

We may also have two infinitely long sequences over some symbol alphabet. Again, the Hamming distance is defined as the number of places in which the two sequences are different. The Hamming distance between two infinite sequences will be infinite unless the sequences are different only on a finite segment.

Suppose that a block codeword is transmitted and a single error is made by the channel in that block. Then the Hamming distance from the senseword to the transmitted codeword is equal to 1. If the distance to every other codeword is larger than 1, then the decoder will properly correct the error if it presumes that the closest codeword to the senseword is the codeword that was actually transmitted.

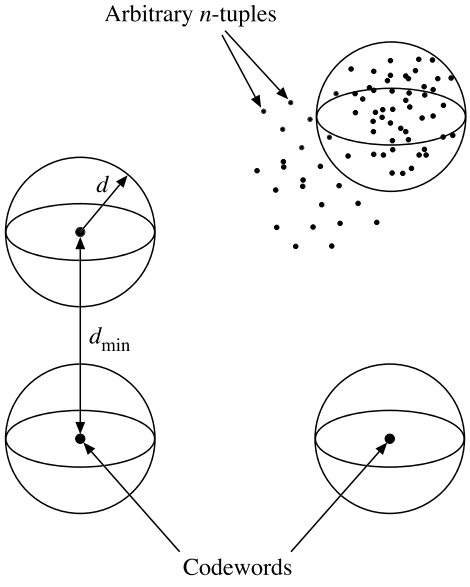


Figure 1.3. Decoding spheres

More generally, if t errors occur, and if the distance from the senseword to every other codeword is larger than t, then the decoder will properly correct the t errors if it presumes that the closest codeword to the senseword was actually transmitted. This always occurs if

$$d_{\min} \geq 2t + 1.$$

It may be possible, sometimes, to correct certain error patterns with t errors even when this inequality is not satisfied. However, correction of t errors cannot be guaranteed if $d_{\min} < 2t + 1$ because then it depends on which codeword is transmitted and on the actual pattern of the t errors within the block.

We shall often describe coding and decoding by using the language of geometry because geometric models are intuitive and powerful aids to reasoning. Figure 1.3 illustrates the geometric situation. Within the space of all q-ary n-tuples, the *Hamming sphere* of radius t (a nonnegative integer), with the center at the sequence $\boldsymbol{v}$, is the set of all sequences $\boldsymbol{v}'$ such that $d(\boldsymbol{v}, \boldsymbol{v}') \leq t$. To define a code within the space of q-ary n-tuples, a set of n-tuples is selected, and these n-tuples are designated as codewords of code $\mathcal{C}$. If $d_{\min}$ is the minimum distance of this code and t is the largest integer satisfying

$$d_{\min} \geq 2t + 1,$$

then nonintersecting Hamming spheres of radius t can be drawn about each of the codewords. A senseword contained in a sphere is decoded as the codeword at the center of that sphere. If t or fewer errors occur, then the senseword is always in the proper sphere, and the decoding is correct.

Some sensewords that have more than t errors will be in a decoding sphere about another codeword and, hence, will be decoded incorrectly. Other sensewords that have more than t errors will lie in the interstitial space between decoding spheres. Depending on the requirements of the application, these can be treated in either of two ways.

A *bounded-distance decoder* decodes only those sensewords lying in one of the decoding spheres about one of the codewords. Other sensewords have more errors than a bounded-distance decoder can correct and are so declared by the decoder. Such error patterns in a bounded-distance decoder are called *uncorrectable error patterns*. When a decoder encounters an uncorrectable error pattern, it declares a *decoding failure*. A bounded-distance decoder is an example of an *incomplete decoder*, which means that it has uncorrectable error patterns. Most error-correcting decoders in use are bounded-distance decoders.

A *complete decoder* decodes every received word into a closest codeword. In geometrical terms, the complete decoder carves up the interstices between spheres and attaches portions to each of the spheres so that each point in an interstice is attached to a closest sphere located nearby. (Some points are equidistant from several spheres and are arbitrarily assigned to one of the closest spheres.) When more than t (but not *too* many) errors occur in a codeword of large blocklength, the complete decoder will usually decode correctly, but occasionally will produce an incorrect codeword. A complete decoder may be preferred for its performance, but for a large code the issue of complexity leads to the use of an incomplete decoder. An incomplete decoder may also be preferred as a way to reduce the probability of decoding error in exchange for a larger probability of decoding failure.

We shall also deal with channels that make *erasures* – or both errors and erasures – as well as channels, called *soft-output channels*, whose output for each symbol is a real number, such as a likelihood measure. A soft-output channel has an input alphabet of size q and an output alphabet consisting of real numbers, or vectors of real numbers. An error-control code can be used with a soft-output channel. The output of the channel then is called a *soft senseword* and the decoder is called a *soft decoder* or a *soft-input decoder*. A soft-input decoder is more tightly interconnected with the modulator and, for this reason, often has very good performance.

For an *erasure channel*, the receiver is designed to declare a symbol erased when that symbol is received ambiguously, as when the receiver recognizes the presence of interference or a transient malfunction. An erasure channel has an input alphabet of size q and an output alphabet of size $q + 1$; the extra symbol is called an *erasure*. For example, an erasure of the third symbol from the message 12345 gives $12 - 45$. This should not be confused with another notion known as a *deletion*, which would give 1245.

An error-control code can be used with an erasure channel. If the code has a minimum distance $d_{\min}$, then any pattern of ρ erasures can be filled if $d_{\min} \geq \rho + 1$. Furthermore,

any pattern of ν errors and ρ erasures can be decoded, provided

$$d_{\min} \geq 2\nu + 1 + \rho$$

is satisfied. To prove this statement, delete the ρ components that contain erasures in the senseword from all codewords of the code. This process gives a new code, called a punctured code, whose minimum distance is not smaller than $d_{\min} - \rho$; hence ν errors can be corrected, provided $d_{\min} - \rho \geq 2\nu + 1$ is satisfied. In this way we can recover the punctured codeword, which is equivalent to the original codeword with ρ components erased. Finally, because $d_{\min} \geq \rho + 1$, there is only one codeword that agrees with the unerased components; thus the entire codeword can be recovered.

1.5 Elementary codes

Some codes are simple enough to be described at the outset.

Parity-check codes

These are high-rate codes with poor error performance on a binary output channel. Given k data bits, add a $(k+1)$th bit so that the total number of ones in each codeword is even. Thus for example, with $k = 4$,

$$\begin{array}{cccccccccc} 0 & 0 & 0 & 0 & \leftrightarrow & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \leftrightarrow & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & \leftrightarrow & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & \leftrightarrow & 0 & 0 & 1 & 1 & 0, \end{array}$$

and so forth. This is a $(k+1, k)$ or an $(n, n-1)$ code. The minimum distance is 2, and hence no errors can be corrected. A simple parity-check code is used to detect (but not correct) a single error.

Repetition codes

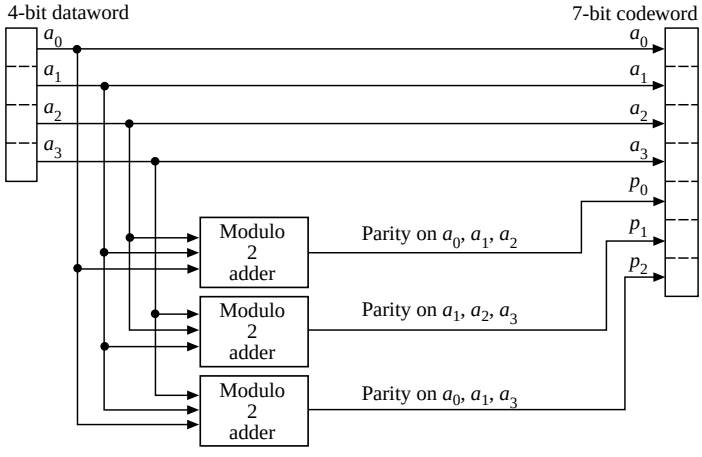
These are low-rate codes with good error performance on a binary output channel. Given a single data bit, repeat it n times. Usually, n is odd

$$\begin{array}{ccccccc} 0 & \leftrightarrow & 0 & 0 & 0 & 0 & 0 \\ 1 & \leftrightarrow & 1 & 1 & 1 & 1 & 1. \end{array}$$

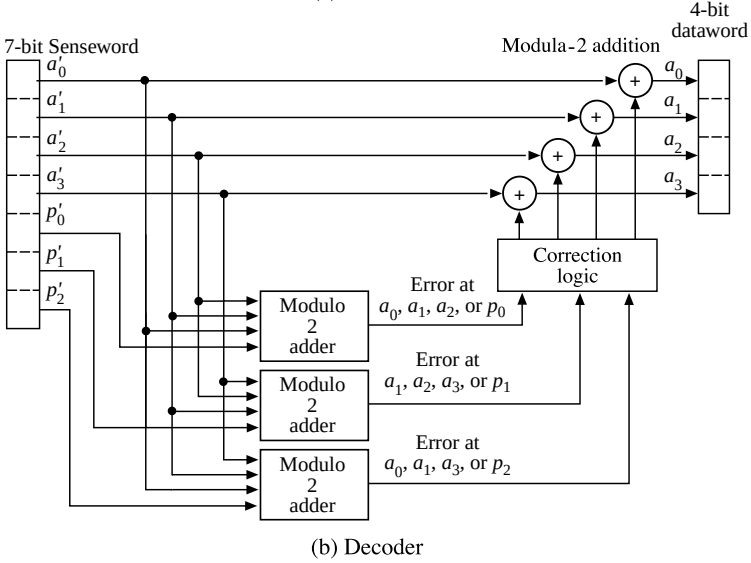
This is an $(n, 1)$ code. The minimum distance is n, and $\frac{1}{2}(n-1)$ errors can be corrected by assuming that the majority of the received bits agrees with the correct data bit.

Hamming codes

These are codes that can correct a single error. For each m, there is a $(2^m - 1, 2^m - 1 - m)$ binary Hamming code. When m is large, the code rate is close to 1, but the fraction of the total number of bits that can be in error is very small. In this section, we will introduce the (7, 4) Hamming codes via a direct descriptive approach. The (7, 4) Hamming code can be described by the implementation in Figure 1.4(a). Given four data bits (a_0, a_1, a_2, a_3), let the first four bits of the codeword equal the four



(a) Encoder



(b) Decoder

Figure 1.4. A simple encoder/decoder for a (7, 4) Hamming code

Table 1.1. *The* (7, 4) *Hamming code*

0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	1	1

data bits. Append three check bits $(p_0,\ p_1,\ p_2)$, defined by

$$p_0 = a_0 + a_1 + a_2$$
$$p_1 = a_1 + a_2 + a_3$$
$$p_2 = a_0 + a_1 + a_3.$$

Here $+$ denotes modulo-2 addition $(0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0)$. The sixteen codewords of the $(7, 4)$ Hamming code are shown in Table 1.1. Of course, the idea of the code is not changed if the bit positions are permuted. All of these variations are equivalent, and all are called the $(7, 4)$ Hamming code.

The decoder receives a seven-bit senseword $\boldsymbol{v} = (a_0', a_1', a_2', a_3', p_0', p_1', p_2')$. This corresponds to a transmitted codeword with at most one error. The decoder, shown in Figure 1.4(b), computes

$$s_0 = p_0' + a_0' + a_1' + a_2'$$
$$s_1 = p_1' + a_1' + a_2' + a_3'$$
$$s_2 = p_2' + a_0' + a_1' + a_3'.$$

The three-bit pattern (s_0, s_1, s_2) is called the *syndrome*. It does not depend on the actual data bits, but only on the error pattern. There are eight possible syndromes: one that corresponds to no error, and one for each of the seven possible patterns with a single error. Inspection shows that each of these error patterns has a unique syndrome, as shown in Table 1.2.

It is a simple matter to design binary logic that will complement the bit location indicated by the syndrome. After correction is complete, the check bits can be discarded.

Table 1.2. *Syndrome table*

Syndrome			Error						
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	1	0	1	0	0	0	0	0

If two or more errors occur, then the design specification of the code is exceeded and the code will miscorrect. That is, it will make a wrong correction and put out incorrect data bits.

Because the (7, 4) Hamming code is a very simple code, it is possible to describe it in this elementary way. A more compact description, which we will eventually prefer, is to use vector space methods, writing the codeword as a vector–matrix product

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ p_0 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

and the syndrome as another matrix–vector product

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \\ p'_0 \\ p'_1 \\ p'_2 \end{bmatrix}.$$

Problems

1.1 a. By trial and error, find a set of four binary words of length 3 such that each word is at least a distance of 2 from every other word.

b. Find a set of sixteen binary words of length 7 such that each word is at least a distance of 3 from every other word.

1.2 a. Describe how to cut 88 circles of 1-inch diameter out of a sheet of paper of width 8.5 inches and length 11 inches. Prove that it is not possible to cut out more than 119 circles of 1-inch diameter.

b. Prove that it is not possible to find 32 binary words, each of length 8 bits, such that every word differs from every other word in at least three places.

1.3 A single-error-correcting Hamming code has $2^m - 1$ bits of which m bits are check bits.

a. Write (n, k) for the first five nontrivial Hamming codes (starting at $m = 3$).

b. Calculate their rates.

c. Write an expression for the probability of decoding error, p_e, when the code is used with a binary channel that makes errors with probability q. How does the probability of error behave with n?

1.4 Design an encoder/decoder for a $(15, 11)$ Hamming code by reasoning as in Figure 1.4. There is no need to show repetitive details (that is, show the principle).

1.5 For any (n, k) block code with minimum distance $2t + 1$ or greater, the number of data symbols satisfies

$$n - k \geq \log_q \left[1 + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t \right].$$

Prove this statement, which is known as the *Hamming bound*.

1.6 The simplest example of a kind of code known as a *product code* is of the form:

a_{00}	a_{01}	$\cdots$	a_{0,k_1-1}	p_{0,k_1}
a_{10}	a_{11}			p_{1,k_1}
$\vdots$			$\vdots$	$\vdots$
$a_{k_2-1,0}$	$\cdots$		a_{k_2-1,k_1-1}	p_{k_2-1,k_1}
$p_{k_2,0}$		$\cdots$	p_{k_2,k_1-1}	p_{k_2,k_1}

where the $k_1 k_2$ symbols in the upper left block are binary data symbols, and each row (and column) is a simple parity-check code. This gives a $((k_1 + 1)(k_2 + 1), k_1 k_2)$ binary product code.

a. Show that p_{k_2,k_1} is a check on both its column and its row.

b. Show that this is a single-error-correcting code.

c. Show that this code is also a double-error-detecting code. Give two double-error patterns that cannot be distinguished from one another when using this code and so cannot be corrected.

d. What is the minimum distance of the code?

1.7 Show that Hamming distance has the following three properties:

(i) $d(\boldsymbol{x}, \boldsymbol{y}) \geq 0$ with equality if and only if $\boldsymbol{x} = \boldsymbol{y}$;

(ii) $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$.

(iii) Triangle inequality

$$d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{y}, \boldsymbol{z}).$$

A distance function with these three properties is called a *metric*.

1.8 a. Show that a code $\mathcal{C}$ is capable of detecting any pattern of d or fewer errors if and only if the minimum distance of the code $\mathcal{C}$ is greater than d.

b. Show that a code is capable of correcting any pattern of t or fewer errors if and only if the minimum distance of the code is at least $2t + 1$.

c. Show that a code can be used to correct all patterns of t or fewer errors and, simultaneously, detect all patterns of d or fewer errors ($d \geq t$) if the minimum distance of the code is at least $t + d + 1$.

d. Show that a code can be used to fill ρ erasures if the minimum distance of the code is at least $\rho + 1$.

1.9 A *soft senseword* $\boldsymbol{v}$ is a vector of real numbers, one corresponding to each bit. To decode a soft senseword, one may choose the codeword $\boldsymbol{c}$ that lies closest to the senseword in euclidean distance

$$d(\boldsymbol{v}, \boldsymbol{c}) = \sum_{i=0}^{n-1} (v_i - c_i)^2.$$

Let $v_i = c_i + e_i$ where the noise components e_i are independent, white, and identical gaussian random variables of variance σ^2 and zero mean. Let $E_m = \sum_{i=0}^{n-1} c_i^2$.

Prove that a binary repetition code using the real numbers ± 1 to represent the code bits has the same energy and the same probability of bit error as an uncoded bit that uses the two real numbers $\pm n$ to represent the value of the single bit.

1.10 a. Show that if the binary (15, 11) Hamming code is used to correct single errors on a channel that makes two errors, the decoder output is always wrong.

b. Show that if the two errors are in check bits, the decoder will always miscorrect a data bit.

c. By appending an overall check bit, show how to extend the (15, 11) Hamming code to a (16, 11) code that corrects all single errors and detects all double errors. What is the minimum distance of this code?

1.11 Show that the list of codewords in Table 1.1 is unchanged by the permutation

$$(c_0, c_1, c_2, c_3, c_4, c_5, c_6) \rightarrow (c_0, c_4, c_1, c_5, c_2, c_6, c_3).$$

Such a permutation is called an automorphism of the code.