Department of Pure Mathematics and Mathematical Statistics
University of Cambridge

CODES AND CRYPTOGRAPHY

The Enigma Cryptography Machine

Notes Lent 2015

T. K. Carne.
t.k.carne@dpmms.cam.ac.uk

Last revised: 10 March, 2015

CONTENTS

1: INTRODUCTION

1.1 Information

This course is about the transmission of information from a Sender to a Receiver. In order to be transmitted via a particular channel, the information needs to be encoded in the correct form, and possibly encrypted if it is to be kept confidential.

The chief purpose of this course is to study information and how it is transmitted from a sender to a receiver. Understanding the amount of information a message contains will tell us how much compression there can be when we encode it. It also gives us a measure of how much duplication of information there is and hence the extent to which we can correct random errors that arise when the message is transmitted.

An *alphabet* $\mathcal{A}$ is a finite set of symbols. For example we might use the usual alphabet:

$$\{a, b, c, d, \ldots, x, y, z, \text{“space”}\}$$

(including a space). Often we use a set of digits $\{0, 1, 2, 3, \ldots, 8, 9\}$ or binary digits $\{0, 1\}$. The symbols from $\mathcal{A}$ are called *letters*. A *message* or *word* is a finite sequence of letters from the alphabet and is usually denoted by juxtaposing the letters:

$$m = a_1 a_2 a_3 \ldots a_{N-1} a_N \ .$$

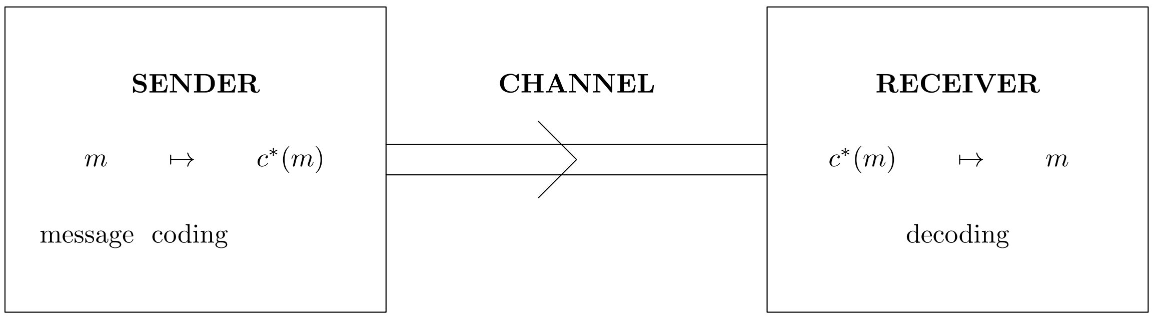
The set of all messages from the alphabet $\mathcal{A}$ is denoted by $\mathcal{A}^*$.

In order to convey the message m successfully, the sender needs to transform it into a form suitable to transmit. For example, if the message is to be recorded digitally, it will need to be transformed into a sequence of binary digits. So the sender *encodes* the message into a new alphabet $\mathcal{B}$. There is a *coding function* $c : \mathcal{A} \to \mathcal{B}^*$ that codes each letter from $\mathcal{A}$ into a finite sequence from $\mathcal{B}$. These finite sequences are called the *code words*. The entire message m is then encoded as

$$c^*(m) = c(a_1)c(a_2)c(a_3)\ldots c(a_{N-1})c(a_N) \in \mathcal{B}^* \ .$$

This is then transmitted via some *channel* to the receiver who needs to decode it to recover the original message m.

The encoding function c may be chosen so that only the intended receiver can decode it. It is then called *encryption*. However, this is by no means necessary. It may be encoded simply to make transmission simpler, or to compress the message to take less space, or to reduce the likelihood of errors being introduced in transmission. We will need to consider all of these.



For example, if we have a message in English $m =$ “Call at 2pm.” that is to be sent in an e-mail. It is first encoded as binary strings using ASCII (American Standard Code for Information Interchange). So $c(C) = 1000011, c(a) = 1100101, \ldots$ and $c^*(m)$ is

1000011 1100101 1101100 1101100 0100000 1100101 1110100 0100000 0110010 1110000 1101101 0101110

(Here the spaces are not part of the encoded message but simply to make the binary string more readable.) This binary string is then transmitted via the internet and decoded by the receiver.

Other examples of coding are:

Morse code

Bar codes

Postcodes

Audio CDs (FLAC:error correcting; MP3: compression)

Mobile 'phone text messages (compression)

Compressing computer files: zip, gz (compression)

Bank Card PINs (cryptography)

* **Digital photographs:** jpeg (compression with data loss)

1.2 Requirements

We will need to use a small amount from a number of earlier courses. In particular it would be helpful if you reviewed:

PROBABILITY	Finite probability spaces, expectation, the weak law of large numbers.
NUMBERS AND SETS	Modulo arithmetic, Euclid's algorithm.
LINEAR ALGEBRA	Vector spaces.
GROUPS, RINGS & MODULES	Finite fields $\mathbb{F}_q$.

In addition, various parts of the course are linked to other courses, notably: Markov Chains; Statistics; Probability and Measure; Linear Analysis.

1.3 Recommended Books

G.M. Goldie & R.G.E. Pinch, **Communication Theory**,
Cambridge University Press 1991.

D. Welsh **Codes and Cryptography**,
Oxford University Press 1988.

T.M. Cover & J.A. Thomas **Elements of Information Theory**,
Wiley 1991.

W. Trappe & L.C. Washington **Introduction to Cryptography with Coding Theory**,
Prentice Hall, 2002.

J.A. Buchmann, **Introduction to Cryptography**, (2nd Ed.)
Springer UTM, 2004

1.4 Example sheets

There will be four example sheets. The first will cover the first two weeks material.

2: ENTROPY

Let $(\mathbb{P}, \Omega)$ be a (discrete) probability on a sample space Ω. For each subset A of Ω, the *information of A* is

$$I(A) = -\log_2 \mathbb{P}(A) \ .$$

Here $\log_2$ is the logarithm to base 2, so $\log_2 p = \ln p / \ln 2$. It is clear that $I(A) \geqslant 0$ with equality if and only if $\mathbb{P}(A) = 1$.

Example:
Let (X_n) be independent Bernoulli random variables that each take the values 0 and 1 with probability $\frac{1}{2}$. For any sequence $x_0 x_1 x_2 \dots x_N$ of N values, the set $A = \{X_1 = x_1, X_2 = x_2, \dots, X_N = x_n\}$ has

$$\mathbb{P}(A) = \left(\frac{1}{2}\right)^N \qquad \text{and so} \qquad I(A) = N \ .$$

Hence the information $I(A)$ measures the number of binary digits (bits) that are specified. Therefore the units for the information are "bits".

Exercise:
Show that two sets $A, B \subset \Omega$ are independent if and only if

$$I(A \cap B) = I(A) + I(B) \ .$$

Let X be a discrete random variable on Ω. For each value x the set $\{X = x\}$ has information

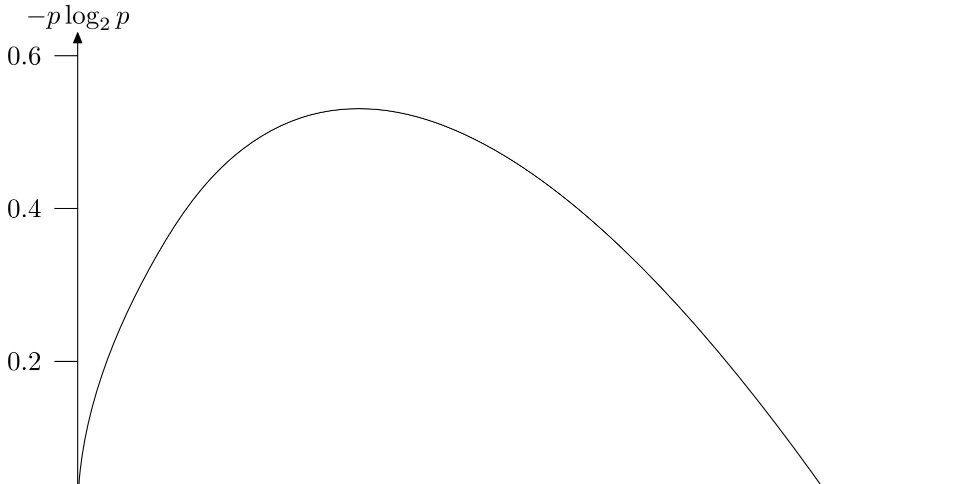
$$I(X = x) = -\log_2 \mathbb{P}(X = x) \ .$$

The expected value of this is the *entropy of X:*

$$H(X) = -\sum_x \mathbb{P}(X = x) \log_2 \mathbb{P}(X = x) \ .$$

This is the average amount of information conveyed by the random variable X. Note that the entropy $H(X)$ does not depend on the particular values taken by X but only on the probability distribution of X, that is on the values $\mathbb{P}(X = x)$.

This definition fails when one of the sets $\{X = x\}$ has probability 0. In this case, note that $h(p) = -p \log_2 p$ tends to 0 as $p \searrow 0$. So we should interpret the term $-\mathbb{P}(X = x) \log_2 \mathbb{P}(X = x)$ as 0 when $\mathbb{P}(X = x) = 0$. The entropy satisfies $H(X) \geqslant 0$, with equality if and only if X is almost surely constant.



Example:
Let X be a random variable and consider a function $f(X)$ of X. Then $H(f(X)) \leqslant H(X)$.

For each value t taken by X set $y = f(t)$, then

$$\mathbb{P}(f(X) = y) = \sum_{x: f(x)=y} \mathbb{P}(X = x) \geqslant \mathbb{P}(X = t) \ .$$

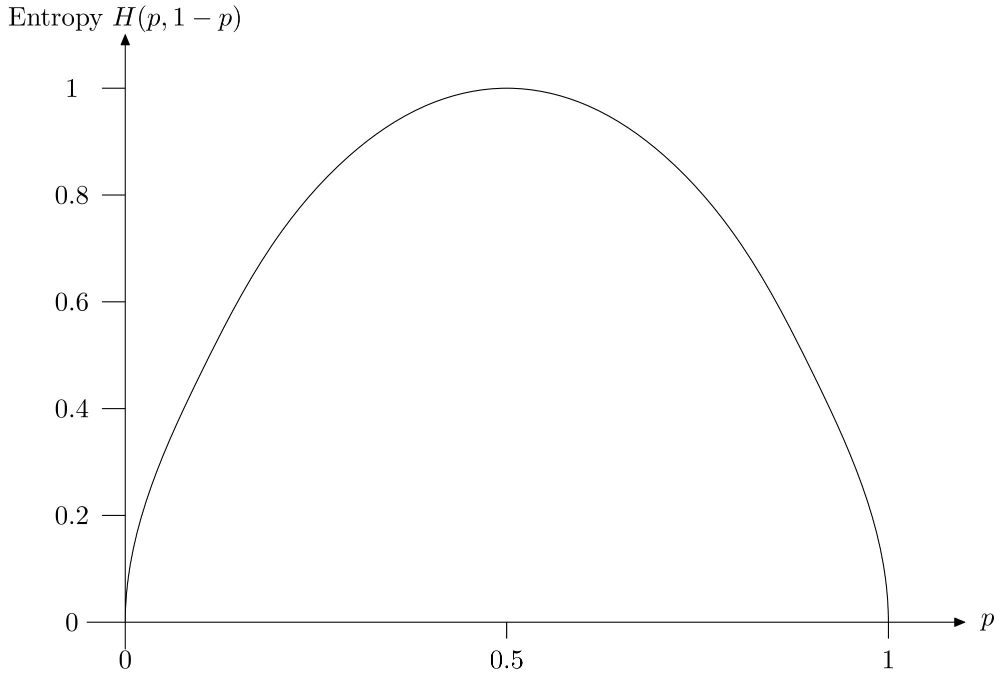
So we have

$$\begin{aligned} H(f(X)) &= -\sum \mathbb{P}(f(X) = y) \log_2 \mathbb{P}(f(X) = y) \\ &= -\sum \mathbb{P}(X = x) \log_2 \mathbb{P}(f(X) = f(x)) \\ &\leqslant -\sum \mathbb{P}(X = x) \log_2 \mathbb{P}(X = x) \ . \end{aligned}$$

Example:
Let $X : \Omega \to \{0, 1\}$ be a Bernoulli random variable that takes the value 1 with probability p and 0 with probability $1 - p$. Then

$$H(X) = -p \log_2 p - (1 - p) \log_2(1 - p) \ .$$

The graph of this as a function of p is concave:



The entropy of X is maximal when $p = \frac{1}{2}$, when it is 1.

The entropy is the average amount of information that we gain by knowing the value of X. We will see later that it is (to within 1) the expected number of yes/no questions we need to ask in order to establish the value of X. We will use the entropy to measure the amount of information in a message and hence to determine how much it can be compressed.

Lemma 2.1 Gibbs' inequality
Let X be a discrete random variable that takes different values with probabilities $(p_k)_{k=1}^K$, so $\sum p_k = 1$. If $(q_k)_{k=1}^K$ is another set of positive numbers with $\sum q_k = 1$ then

$$-\sum_{k=1}^{K} p_k \log_2 p_k \leqslant -\sum_{k=1}^{K} p_k \log_2 q_k \ .$$

There is equality if and only if $q_k = p_k$ for each k.

Proof:

The inequality is equivalent to

$$\sum p_k \ln\left(\frac{q_k}{p_k}\right) \leqslant 0 \ .$$

Now the natural logarithm function is concave, so its graph lies below the tangent line at the point $(1, 0)$, that is

$$\ln t \leqslant t - 1 \ .$$

Indeed, the natural logarithm is strictly concave since its second derivative is strictly negative. Hence $\ln t < t - 1$ unless $t = 1$. Therefore,

$$\sum p_k \ln\left(\frac{q_k}{p_k}\right) \leqslant \sum p_k \left(\frac{q_k}{p_k} - 1\right) = \sum (q_k - p_k) = 0$$

with equality if and only if $q_k/p_k = 1$ for each k. □

Example:
A random variable Y that takes K distinct values with equal probability $1/K$ has entropy $\log_2 K$.

If X is any random variable that takes K different values then its entropy is at most $\log_2 K$.

Thus the entropy is maximal when the values taken by X are all equally likely.
(You should compare this with the results in thermodynamics that entropy is maximal when states are all equally likely.)

Let X, Y be two random variables, then (X, Y) is also a (vector-valued) random variable with entropy $H(X, Y)$. This is called the *joint entropy of X and Y*. Gibbs' inequality (2.1) enables us to prove:

Corollary 2.2
Let X, Y be discrete random variables. Then the joint entropy satisfies

$$H(X, Y) \leqslant H(X) + H(Y)$$

with equality if and only if X and Y are independent.

Proof:

The joint entropy is

$$H(X, Y) = -\sum \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x, Y = y) \ .$$

The products $\mathbb{P}(X = x)\mathbb{P}(Y = y)$ sum to 1 and so Gibbs' inequality gives

$$\begin{aligned} H(X, Y) &= -\sum \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x, Y = y) \\ &\leqslant -\sum \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x)\mathbb{P}(Y = y) \\ &= -\sum \mathbb{P}(X = x, Y = y) \left(\log_2 \mathbb{P}(X = x) + \log_2 \mathbb{P}(Y = y)\right) \\ &= -\sum \mathbb{P}(X = x) \log_2 \mathbb{P}(X = x) - \sum \mathbb{P}(Y = y) \log_2 \mathbb{P}(Y = y) \end{aligned}$$

with equality if and only if

$$\mathbb{P}(X = x, Y = y) = \mathbb{P}(X = x)\mathbb{P}(Y = y) \qquad \text{for all } x, y \ .$$

This is the condition that X and Y are independent random variables. □

Example:
Let X be a random variable that takes D values each with probability $1/D$. Then X has entropy $\log_2 D$. Now suppose that $X_1, X_2, \dots, X_N$ are independent random variables each with the same distribution as X. Then the entropy of the sequence $(X_1, X_2, \dots, X_N)$ is $N \log_2 D$.

We can refine the last corollary by considering conditional distributions. If x is a value taken by the random variable X with non-zero probability, then the conditional distribution of another random variable Y given $X = x$ is

$$\mathbb{P}(Y = y | X = x) = \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(X = x)} \ .$$

This gives a random variable $(Y|X = x)$. Now observe that

$$\begin{aligned} H(X,Y) &= -\sum \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x, Y = y) \\ H(X) &= -\sum \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x) \end{aligned}$$

so

$$\begin{aligned} H(X,Y) - H(X) &= -\sum \mathbb{P}(X = x, Y = y) \log_2 \frac{\mathbb{P}(X = x, Y = y)}{\mathbb{P}(X = x)} \\ &= -\sum \mathbb{P}(X = x)\mathbb{P}(Y = y|X = x) \log_2 \mathbb{P}(Y = y|X = x) \\ &= \sum \mathbb{P}(X = x) H(Y|X = x) \ . \end{aligned}$$

This is called the *conditional entropy* $H(Y|X)$, so

$$H(Y|X) = H(X,Y) - H(X) \ .$$

Now observe that, for each value x, we have

$$H(Y|X = x) \geqslant 0$$

with equality if and only if $(Y|X = x)$ is almost surely constant. Therefore,

$$H(X,Y) - H(X) = H(Y|X) \geqslant 0$$

with equality if and only if Y is almost surely a function of X. These results show that

$$H(X) + H(Y) \geqslant H(X,Y) \geqslant H(X) \ .$$

3: CODING

3.1 Prefix-free Codes

Suppose that we encode a message $m = a_1a_2a_3\dots a_N$ using the function $c : \mathcal{A} \to \mathcal{B}^*$. Each letter a in the alphabet $\mathcal{A}$ is replaced by the code word $c(a)$, which is a finite sequence of letters from the alphabet $\mathcal{B}$. The entire message m is coded as

$$c^*(m) = c(a_1)c(a_2)c(a_3)\dots c(a_N)\ .$$

Here c^* is the induced map $c^* : \mathcal{A}^* \to \mathcal{B}^*$. To be useful, it must be possible to decode $c^*(m)$ and recover the original message m. In this case we say that c is *decodable* (or *decipherable*). This means that c^* must be injective.

In order for c to be decodable, the coding function $c : \mathcal{A} \to \mathcal{B}^*$ must certainly be injective. However, this is not sufficient. For example, the code c that sends the nth letter to a string of n 0s is injective but c^* is not injective since every coded message is a string of 0s.

Let w be a word in $\mathcal{B}^*$, that is a finite sequence of letters from $\mathcal{B}$. Another word w' is a *prefix of* w if w is the concatenation of w' with another word w'', so $w = w'w''$. A code $c : \mathcal{A} \to \mathcal{B}^*$ is *prefix-free* if no code word $c(a)$ is a prefix of a different code word $c(a')$.

Example:
The code

$$c : \{0, 1, 2, 3\} \to \{0, 1\}^*$$

which maps

$$\begin{aligned} c &: 0 \mapsto 0 \\ c &: 1 \mapsto 10 \\ c &: 2 \mapsto 110 \\ c &: 3 \mapsto 111 \end{aligned}$$

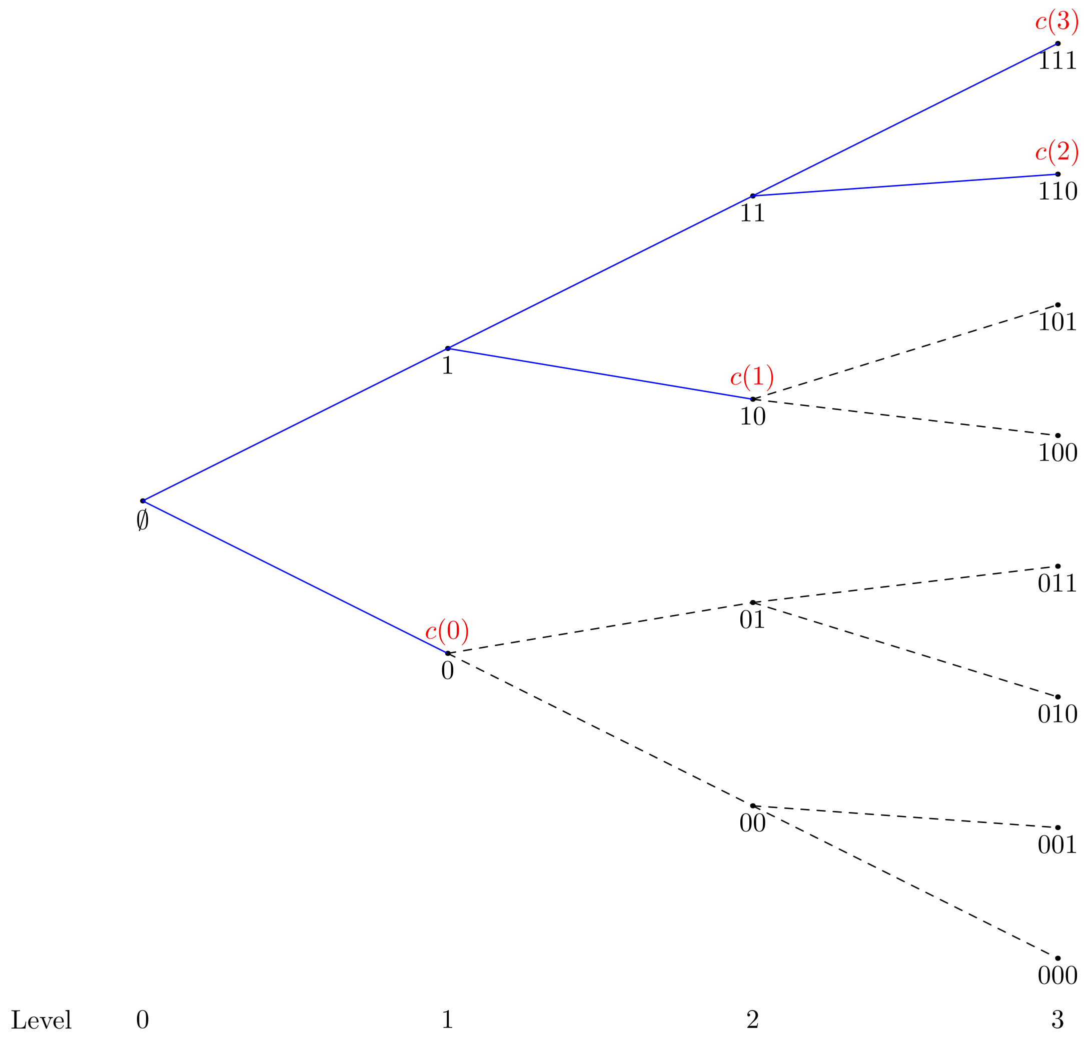
is prefix-free. However, if we reversed each of the code words we would not have a prefix-free code but it would still be decadable.

Any prefix-free code is decodable. For, if we receive a message $w \in \mathcal{B}^*$ we can decode it by looking at the prefixes of w. One of these must be $c(a_1)$ for some letter $a_1 \in \mathcal{A}$. Since the code is prefix-free, this letter a_1 is unique. Now delete this prefix and repeat the process to find a_2 *etc.* Not only can we decode the message but we can decode it as it is received: We do not need to wait for the end of the message to decode the first letter. As soon as we have received all of $c(a_n)$ we can tell what a_n was. For this reason, prefix-free codes are sometimes called *instantaneous* codes or *self-punctuating* codes. They are especially convenient and we will almost always use them.

We can also think about prefix-free codes in terms of trees. For the alphabet $\mathcal{B}$ draw a tree with a base vertex at the empty string $\emptyset$ and vertices labelled by words in $\mathcal{B}^*$. Each vertex w is joined to the vertices wb for each letter $b \in \mathcal{B}$. The prefixes of a word w are then all of the vertices of the tree on the path from the base vertex $\emptyset$ to w. So, a code is prefix-free if the path from any code word to the base vertex contains no other code word. This is a useful way to think about prefix-free codes.

The *vertices at level* N in this tree are those N steps from the base vertex, so those labelled by words of length N.

Example:
The tree for the example code above is:



3.2 Kraft's Inequality

Proposition 3.1 Kraft's Inequality I
Let $c : \mathcal{A} \to \mathcal{B}^*$ be a prefix-free code with the code word $c(a)$ having length $l(a)$. Then

$$\sum_{a \in \mathcal{A}} D^{-l(a)} \leqslant 1 \tag{$*$}$$

where $D = |\mathcal{B}|$ is the number of letters in the alphabet $\mathcal{B}$.

Proof:
Let $L = \max\{l(a) : a \in \mathcal{A}\}$. Consider the tree constructed from the prefix-free code c. The vertices at level L are those corresponding to words of length L, so there are D^L such vertices.

A code word $c(a)$ with length $l(a)$ is a prefix of $D^{L-l(a)}$ such vertices. Moreover, since c is prefix-free, no level L vertex can have two code words as a prefix. Hence, the total number of level L vertices with code words as prefixes is

$$\sum_{a \in \mathcal{A}} D^{L-l(a)}$$

and this can be at most the total number of level L vertices: D^L. So

$$\sum_{a\in\mathcal{A}} D^{-l(a)} \leqslant 1$$

as required. □

The converse to this is also true.

Proposition 3.2 Kraft's Inequality II
Let $\mathcal{A}, \mathcal{B}$ be finite alphabets with $\mathcal{B}$ having D letters. If $l(a)$ are positive integers for each letter $a \in \mathcal{A}$, satisfying

$$\sum_{a\in\mathcal{A}} D^{-l(a)} \leqslant 1$$

then there is a prefix-free code $c : \mathcal{A} \to \mathcal{B}^*$ with each code word $c(a)$ having length $l(a)$.

Proof:
First re-order the letters in $\mathcal{A}$ as $a_1, a_2, \dots, a_K$ so that the lengths $l_j = l(a_j)$ satisfy

$$l_1 \leqslant l_2 \leqslant l_3 \leqslant \dots \leqslant l_K \ .$$

We will define the code c inductively.

Choose any word of length l_1 as $c(a_1)$.

Suppose that $c(a_j)$ has been chosen for $j < k$ so that no $c(a_i)$ is a prefix of $c(a_j)$ for $i < j < k$. Consider the words of length l_k. There are D^{l_k} of these. Of these, $D^{l_k - l_j}$ have $c(a_j)$ as a prefix. However, the given inequality shows that

$$1 + \sum_{j=1}^{k-1} D^{l_k - l_j} = \sum_{j=1}^{k} D^{l_k - l_j} \leqslant D^{l_k} \ .$$

So we can choose a word $c(a_k)$ of length l_k which does not have any $c(a_j)$ (for $j < k$) as a prefix.

By induction, we can construct the required prefix-free code. □

Note that the code constructed above is far from unique.

Kraft's inequality was strengthened by McMillan who showed that it holds for any decodable code, even if it is not prefix-free. This means that if we are concerned only with the lengths of the code words we need only ever consider prefix-free codes.

Proposition 3.3 McMillan
Let $c : \mathcal{A} \to \mathcal{B}^*$ be a decodable code with the code word $c(a)$ having length $l(a)$. Then

$$\sum_{a\in\mathcal{A}} D^{-l(a)} \leqslant 1 \qquad (*)$$

where $D = |\mathcal{B}|$ is the number of letters in the alphabet $\mathcal{B}$.

Proof:
Let L be the maximum length of code word $c(a)$.

Consider the power

$$\left(\sum_{a \in \mathcal{A}} D^{-l(a)} \right)^R$$

for a natural number R. If we expand the bracket we obtain an expression of the form:

$$\sum D^{-(l(a_1)+l(a_2)+\ldots+l(a_R))}$$

where the sum is over all sequences $a_1, a_2, a_3, \ldots, a_R$ of length R. Now the word

$$w = c(a_1)c(a_2)c(a_3)\ldots c(a_R) \in \mathcal{B}^*$$

has length

$$|w| = l(a_1) + l(a_2) + \ldots + l(a_R) \leqslant RL \ .$$

Since c is decodable, every word $w \in \mathcal{B}^*$ with length $|w| = m$ can come from at most one sequence $a_1 a_2 \ldots a_R$. Hence,

$$\left(\sum_{a \in \mathcal{A}} D^{-l(a)} \right)^R \leqslant \sum_{m=1}^{RL} n(m) D^{-m}$$

where $n(m)$ is the number of words w of length m that are of the form $c(a_1)c(a_2)c(a_3)\ldots c(a_R)$. The total number of words of length m is at most D^m, so $n(m) \leqslant D^m$. Therefore

$$\left(\sum_{a \in \mathcal{A}} D^{-l(a)} \right)^R \leqslant \sum_{m=1}^{RL} D^m D^{-m} = RL \ .$$

Taking Rth roots gives

$$\sum_{a \in \mathcal{A}} D^{-l(a)} \leqslant (RL)^{1/R}$$

and taking the limit as $R \to \infty$ gives the desired result. $\square$

4: EFFICIENT CODES

There is often a cost for each letter transmitted through a channel, either a financial cost or a cost in terms of space or time required. So we try to find codes where the expected length of code words is as small as possible. We will call such a code optimal.

4.1 Shannon – Fano Codes

Let $c : \mathcal{A} \to \mathcal{B}^*$ be a code. We will assume that this code is decodable and, indeed, prefix-free. For each letter $a \in \mathcal{A}$, the code word $c(a)$ has length $|c(a)|$. We wish to find codes where the expected value of this length is as small as it can be. To find an expected value we need to fix a probability distribution on $\mathcal{A}$. Let $p(a)$ be the probability of choosing the letter $a \in \mathcal{A}$. Then the *expected length of a code word* is

$$\sum_{a \in \mathcal{A}} p(a)|c(a)| \ .$$

It is convenient to let A be a random variable that takes values in $\mathcal{A}$ with $\mathbb{P}(A = a) = p(a)$. Then the expected length of a code word is

$$\mathbb{E}|c(A)| \ .$$

Kraft's inequality (3.1) shows that the lengths of the code words $l(a) = |c(a)|$ must satisfy

$$\sum D^{-l(a)} \leqslant 1 \ . \tag{$*$}$$

Moreover, Proposition 3.2 shows that if we have any positive integers $l(a)$ which satisfy $(*)$, then there is a prefix-free code with code word lengths $l(a)$. Hence we need to solve the optimisation problem:

$$\text{Minimize } \sum p(a)l(a) \ \text{ subject to } \ \sum D^{-l(a)} \leqslant 1 \ \text{ and each } \ l(a) \ \text{ an integer.}$$

It is informative to first solve the simpler optimization problem where we do not insist that the values $l(a)$ are all integers.

Example:
Show that the minimum value of $\sum p(a)l(a)$ subject to $\sum D^{-l(a)} \leqslant 1$ occurs when

$$l(a) = -\log_D p(a) = -\frac{\log_2 p(a)}{\log_2 D} \ .$$

If we define $l(a)$ by this formula then we certainly have

$$\sum D^{-l(a)} = \sum p(a) = 1$$

and the expected length of code words becomes

$$\sum p(a)l(a) = \frac{-\sum p(a)\log_2 p(a)}{\log_2 D} = \frac{H(A)}{\log_2 D} \ .$$

We need to show that for any values $l(a)$ that satisfy $\sum D^{-l(a)} \leqslant 1$ we have

$$\frac{-\sum p(a)\log_2 p(a)}{\log_2 D} \leqslant \sum p(a)l(a) \ .$$

Set $S = \sum D^{-l(a)}$ and $q(a) = D^{-l(a)}/S$. Then $\sum q(a) = 1$ and Gibbs' inequality (2.1) shows that

$$\begin{aligned} -\sum p(a)\log_2 p(a) \leqslant -\sum p(a)\log_2 q(a) &= \sum p(a)\left(l(a)\log_2 D + \log_2 S\right) \\ &= \left(\sum p(a)l(a)\right)\log_2 D + \log_2 S \leqslant \left(\sum p(a)l(a)\right)\log_2 D \end{aligned}$$

as required.

Proposition 4.1
Let A be a random variable that takes values in a finite alphabet $\mathcal{A}$. For any decodable code $c : \mathcal{A} \to \mathcal{B}^*$ the expected length of code words satisfies

$$\mathbb{E}|c(A)| \geqslant \frac{H(A)}{\log_2 D}$$

where D is the number of letters in $\mathcal{B}$.

Proof:

Kraft's inequality (3.1) shows that the lengths $l(a) = |c(a)|$ must satisfy $\sum D^{-l(a)} \leqslant 1$. Hence, we see as in the example above, that

$$\sum p(a)l(a) \geqslant \frac{H(A)}{\log_2 D}$$

as required. $\square$

We can not always find a code that achieves equality in Proposition 4.1. However, a very simple argument shows that we can get close.

Proposition 4.2 Shannon – Fano encoding
Let A be a random variable that takes values in a finite alphabet $\mathcal{A}$. There is a prefix-free code $c : \mathcal{A} \to \mathcal{B}^*$ for which the expected length of code words satisfies

$$\mathbb{E}|c(A)| < 1 + \frac{H(A)}{\log_2 D}$$

where D is the number of letters in $\mathcal{B}$.

Proof:

Kraft's inequality shows that we only need to find integers $l(a)$ that satisfy

$$\sum p(a)l(a) < 1 + \frac{H(A)}{\log_2 D} \qquad \text{and} \qquad \sum D^{-l(a)} \leqslant 1 \ .$$

If the lengths $l(a)$ are to be integers, then we can not always take $l(a) = -\log_D p(a)$. However, we can set

$$l(a) = \lceil -\log_D p(a) \rceil \ .$$

For this choice of $l(a)$ we have

$$-\log_D p(a) \leqslant l(a) \qquad \text{so} \qquad p(a) \geqslant D^{-l(a)} \ .$$

This certainly implies that $\sum D^{-l(a)} \leqslant 1$, so there is a prefix-free code with these word lengths. Moreover,

$$\sum p(a)l(a) < \sum p(a)\left(1 - \log_D p(a)\right) = 1 - \sum p(a) \log_D p(a) = 1 + \frac{H(A)}{\log_2 D} \ .$$

Given the word lengths $l(a)$, we can now use Proposition 3.2 to construct the desired code c. $\square$

The code produced in this way is called a *Shannon – Fano* code. It is not quite optimal but is easy to construct and has expected word lengths very close to the optimal value.

Example:
The elements of the alphabet $\mathcal{A} = \{0, 1, 2, 3\}$ are taken with probabilities $0.4, 0.3, 0.2, 0.1$. So we find:

a	$p(a)$	$-\log_2 p(a)$	$\lceil -\log_2 p(a) \rceil$
0	0.4	1.32	2
1	0.3	1.74	2
2	0.2	2.32	3
3	0.1	3.32	4

Hence we see that a Shannon – Fano code is

$$\begin{aligned} c :& 0 \mapsto 00 \\ c :& 1 \mapsto 01 \\ c :& 2 \mapsto 100 \\ c :& 3 \mapsto 1100 \end{aligned}$$

This code has $\mathbb{E}|c(A)| = 2.4$ while the entropy is $H(A) = 1.85$. So we have, as expected,

$$H(A) \leqslant \mathbb{E}|c(A)| < H(A) + 1 \ .$$

This code does not achieve the smallest possible value for $\mathbb{E}|c(A)|$. For the code

$$\begin{aligned} h :& 0 \mapsto 0 \\ h :& 1 \mapsto 10 \\ h :& 2 \mapsto 110 \\ h :& 3 \mapsto 111 \end{aligned}$$

has $\mathbb{E}|h(A)| = 1.9$. (This is indeed the optimal value and h is a Huffman code.)

The two propositions 4.1 and 4.2 together give:

Theorem 4.3 Shannon's noiseless coding theorem
Let A be a random variable that takes values in a finite alphabet $\mathcal{A}$. An optimal code $c : \mathcal{A} \to \mathcal{B}^*$ for an alphabet $\mathcal{B}$ of size D satisfies

$$\frac{H(A)}{\log_2 D} \ \leqslant \ \mathbb{E}|c(A)| \ < \ \frac{H(A)}{\log_2 D} + 1 \ .$$

Example:
We can ask any questions with a yes or no answer about the value taken by the random variable A. What is the average number of questions we need to ask to determine the value of A?

Each question gives a map

$$\mathcal{A} \to \{0, 1\} \ ; a \mapsto \begin{cases} 1 & \text{if the answer is yes for } a; \\ 0 & \text{if the answer is no for } a. \end{cases}$$

The sequence of these questions thus gives a code $c : \mathcal{A} \to \{0, 1\}^*$. The sequence of answers determines the value of a if and only if this code is decodable.

So Shannon's noiseless coding theorem shows that it is possible to choose the questions with

$$H(A) \ \leqslant \ \mathbb{E}|c(A)| \ < \ H(A) + 1 \ .$$

Thus the average number of questions required is within 1 of the entropy $H(A)$.

4.2 Huffman Codes

We need to work a little harder to obtain a code that is actually optimal. For simplicity we will only do this in the case where $\mathcal{B}$ is the binary alphabet $\{0,1\}$. Huffman gave an inductive definition of codes that we can prove are optimal. When Huffman was a graduate student his teacher, Fano, offered students a chance to avoid the examination at the end of the course by finding an algorithm for optimal codes. Huffman duly did this.

Let $\mathcal{A}$ be a finite alphabet with probabilities $p(a)$. We will label the elements of $\mathcal{A}$ as $a_1, a_2, \dots, a_K$ so that the probabilities $p_k = p(a_k)$ satisfy

$$p_1 \geqslant p_2 \geqslant p_3 \geqslant \dots \geqslant p_K \ .$$

When $K = 2$ the Huffman code is simply

$$h : \{a_1, a_2\} \to \{0,1\} \ ; \quad h(a_1) = 0 \ , h(a_2) = 1 \ .$$

This is clearly optimal. Suppose that a Huffman code has been defined for alphabets of size $K-1$. Choose the two letters in $\mathcal{A}$ that have the smallest probabilities: p_{K-1} and p_K. Form a new alphabet $\widetilde{\mathcal{A}} = \{a_1, a_2, \dots, a_{K-2}, a_{K-1} \cup a_K\}$ by combining the letters a_{K-1} and a_K to give a new letter $a_{K-1} \cup a_K$ in $\widetilde{\mathcal{A}}$ that is given the probability $p_{K-1} + p_K$. Let $\widetilde{h} : \widetilde{\mathcal{A}} \to \{0,1\}^*$ be a Huffman code for this new alphabet. Then the Huffman code for $\mathcal{A}$ is obtained by adding a 0 or a 1 to the end of the $\widetilde{h}$ code words for a_{K-1} and a_K:

$$h(a_j) = \begin{cases} \widetilde{h}(a_j) & \text{for } j = 1, 2, \dots, K-2; \\ \widetilde{h}(a_{K-1} \cup a_K)0 & \text{for } j = K-1; \\ \widetilde{h}(a_{K-1} \cup a_K)1 & \text{for } j = K \ . \end{cases}$$

This defines the Huffman code. It is clearly prefix-free.

Example:
Consider the alphabet $\mathcal{A} = \{0,1,2,3,4\}$ with probabilities $0.4, 0.2, 0.15, 0.15, 0.1$. The letters with the smallest probabilities are 3 and 4, so combine these to form $3 \cup 4$. This gives $\{0, 1, 2, 3 \cup 4\}$ with probabilities $0.4, 0.2, 0.15, 0.25$.

The letters with the smallest probabilities are now 1 and 2, so combine these to form $1 \cup 2$. This gives $0, 1 \cup 2, 3 \cup 4$ with probabilities $0.4, 0.35, 0.25$.

The letters with the smallest probabilities are now $1 \cup 2, 3 \cup 4$, so combine these to form $(1 \cup 2) \cup (3 \cup 4)$ with probability 0.6.

Hence we see that a Huffman code is:

$$\begin{aligned} h &: 0 \mapsto 0 \\ h &: 1 \mapsto 100 \\ h &: 2 \mapsto 101 \\ h &: 3 \mapsto 110 \\ h &: 4 \mapsto 111 \end{aligned}$$

This has expected code length $\mathbb{E}|h(A)| = 0.4 + 3 \times 0.6 = 2.2$ compared with an entropy $H(A) = 2.15$.

What is the Shannon – Fano code? Is it optimal?

We will now prove that Huffman codes are optimal.

Theorem 4.4 Huffman codes are optimal
A Huffman code is optimal. That is, its average length of code words is as small as possible for any decodable code from the alphabet $\mathcal{A}$ into $\{0,1\}^*$.

As above, we label the letters in $\mathcal{A}$ as $a_1, a_2, \dots, a_K$ with probabilities $p_1 \geqslant p_2 \geqslant p_3 \geqslant \dots \geqslant p_K > 0$. The random variable A takes the value a_j with probability p_j. So the average code length is $\mathbb{E}|c(A)|$.

Lemma 4.5 Optimal codes
There is an optimal code $c : \mathcal{A} \to \{0,1\}^*$ that satisfies:

(a) *The lengths $l_j = |c(a_j)|$ are ordered inversely to the probabilities, so $l_1 \leqslant l_2 \leqslant \dots \leqslant l_K$.*

(b) *The code words $c(a_{K-1})$ and $c(a_K)$ have the same length and differ only in the last bit.*

Proof:
We know that there is some prefix-free code for $\mathcal{A}$. Let its expected code length be C. Then there are only a finite number of codes c with $\sum p_k l_k \leqslant C$. So there must be a prefix-free code that achieves the minimum value for $\sum p_k l_k$. This is an optimal code.

(a) If $p_i \geqslant p_j$ but $l_i > l_j$, then we could reduce $\sum p_k l_k$ by interchanging the code words $c(a_i)$ and $c(a_j)$. Hence we must have $l_1 \leqslant l_2 \leqslant l_3 \leqslant \dots \leqslant l_K$.

(b) Let L be the maximum code word length, so $L = l_K$. Write the code word $c(a_K)$ as wb where w is a word of length $L-1$ and b is the final bit. Consider the new code we obtain by changing the single code word $c(a_K)$ to w. This reduces the length and so reduces $\sum p_j l_j$. Since c was optimal, this new code can not be prefix-free. Hence there must be another code word $c(a_j)$ that has w as a prefix. This means that $c(a_j)$ and $c(a_K)$ are both of length L and differ only in their last bit.

Permute the code words of length L so that $c(a_{K-1})$ and $c(a_K)$ differ only in their last bit. □

We now return to the proof of Theorem 4.4: Huffman codes are optimal. We will prove this by induction on the size K of $\mathcal{A}$. The result is clear for $K = 2$. Assume that the result is true for alphabets with size $K-1$.

Let $h : \mathcal{A} \to \{0,1\}^*$ be a Huffman code and $c : \mathcal{A} \to \{0,1\}^*$ an optimal code. The construction of the Huffman code shows that there is a Huffman code $\widetilde{h}$ on the new alphabet

$$\widetilde{\mathcal{A}} = \{a_1, a_2, \dots, a_{K-2}, a_{K-1} \cup a_K\}$$

of size $K-1$. The letters here have probabilities $p_1, p_2, \dots, p_{K-2}, p_{K-1} + p_K$ respectively and we will write $\widetilde{A}$ for a random variable that takes the letters in $\widetilde{\mathcal{A}}$ with these probabilities.. The code h is then given by

$$h(a_j) = \begin{cases} \widetilde{h}(a_{K-1} \cup a_K)0 & \text{if } j = K-1; \\ \widetilde{h}(a_{K-1} \cup a_K)1 & \text{if } j = K; \\ \widetilde{h}(a_j) & \text{if } j \leqslant K-2. \end{cases}$$

This means that the expected code length for h and $\widetilde{h}$ satisfy

$$\mathbb{E}|h(A)| = \mathbb{E}|\widetilde{h}(\widetilde{A})| \ + \ (p_{K-1} + p_K) \ .$$

For the optimal code c, we apply the lemma (4.5) . So choose c with $c(a_{K-1})$ and $c(a_K)$ differing only in their last bit, say $c(a_{K-1}) = w0$ and $c(a_K) = w1$ for a word $w \in \{0,1\}^*$. Define a new code on $\widetilde{A}$ by setting $\widetilde{c}(a_{K-1} \cup a_K) = w$ and $\widetilde{c}(a_j) = c(a_j)$ for $j \leqslant K-2$. This is readily seen to be prefix-free and has

$$\mathbb{E}|c(A)| = \mathbb{E}|\widetilde{c}(\widetilde{A})| \ + \ (p_{K-1} + p_K) \ .$$

The inductive hypothesis shows that $\widetilde{h}$ is optimal. So

$$\mathbb{E}|\widetilde{h}(\widetilde{A})| \leqslant \mathbb{E}|\widetilde{c}(\widetilde{A})| \ .$$

Therefore,

$$\mathbb{E}|h(A)| \leqslant \mathbb{E}|c(A)| \ .$$

However, $\mathbb{E}|c(A)|$ is the minimal average code length, so we must have equality and hence h is itself optimal. □

5: COMPRESSION

Throughout this section $\mathcal{A}$ will be an alphabet of K letters. We wish to send messages $m = a_1a_2a_3\dots$ with each $a_j \in \mathcal{A}$. We will let A_j be a random variable that gives the jth letter and we will assume that each A_j is identically distributed with $\mathbb{P}(A_j = a) = p(a)$. We wish to encode the message by strings in an alphabet $\mathcal{B}$ of size $D = |\mathcal{B}|$.

5.1 Block Codes

In the last section we considered a code $c : \mathcal{A} \to \mathcal{B}^*$. The expected code length for this code is

$$\mathbb{E}|c(A_1)| = \sum p(a)|c(a)| \ .$$

The noiseless coding theorem (4.3) showed that there was an optimal code c for which

$$\frac{H(A_1)}{\log_2 D} \leqslant \mathbb{E}|c(A_1)| < 1 + \frac{H(A_1)}{\log_2 D} \ .$$

However, rather than encoding the message one letter at a time we could divide it into blocks and encode each block. Divide the message m into blocks of length r:

$$m = (a_1a_2\dots a_r)(a_{r+1}a_{r+2}\dots a_{2r})(a_{2r+1}a_{2r+2}\dots a_{3r})\dots$$

Each block is in the new alphabet $\mathcal{A}^r$ and we can find an optimal code for this:

$$c_r : \mathcal{A}^r \to \mathcal{B}^* \qquad \text{with} \qquad \frac{H(A_1, A_2, \dots, A_r)}{\log_2 D} \leqslant \mathbb{E}|c_r(A_1, A_2, \dots, A_r)| < 1 + \frac{H(A_1, A_2, \dots, A_r)}{\log_2 D} \ .$$

We call this a *block code*. Of course, such a block code deals with r letters of $\mathcal{A}$ at a time and so the average code length **per letter** is $\dfrac{\mathbb{E}|c_r(A_1, A_2, \dots, A_r)|}{r}$. This satisfies

$$\frac{H(A_1, A_2, \dots, A_r)}{r\log_2 D} \leqslant \frac{\mathbb{E}|c_r(A_1, A_2, \dots, A_r)|}{r} < \frac{1}{r} + \frac{H(A_1, A_2, \dots, A_r)}{r\log_2 D} \ .$$

Thus the expected code length per letter is approximately $\dfrac{H(A_1, A_2, \dots, A_r)}{r\log_2 D}$.

In Corollary 2.2 we saw that

$$H(A_1, A_2, \dots, A_r) \leqslant H(A_1) + H(A_2) + \dots + H(A_r)$$

with equality when the letters $A_1, A_2, \dots$ are independent. The random variables (A_j) all have the same distribution so we see that

$$H(A_1, A_2, \dots, A_r) \leqslant rH(A_1) \ .$$

Consider first the case where the letters (A_j) are independent. Then $H(A_1, A_2, \dots, A_r) = rH(A_1)$ and so we see that the optimal block code satisfies

$$\frac{H(A_1)}{\log_2 D} \leqslant \frac{\mathbb{E}|c_r(A_1, A_2, \dots, A_r)|}{r} < \frac{1}{r} + \frac{H(A_1)}{\log_2 D} \ .$$

So the expected code length per letter tends to $H(A_1)/\log_2 D$ as the block length r increases to infinity.

We can also deal with more complicated cases where the letters (A_j) are not independent. In English the letters are far from independent; for example the letter following q is very likely to be u.

As a simple model we will assume that the probability that A_j takes a value a_j depends only on the previous letter A_{j-1}, so

$$\mathbb{P}(A_j = a_j | a_1 = a_1, A_2 = a_2, \ldots, A_{j-1} = a_{j-1}) = \mathbb{P}(A_j = a_j | A_{j-1} = a_{j-1})$$

and that this is independent of j. This implies that

$$H(A_j | A_1, A_2, \ldots, A_{j-1}) = H(A_j | A_{j-1}) = H(A_2 | A_1) \ .$$

Now we know that

$$H(A_1, A_2, \ldots, A_r) = H(A_r | A_1, A_2, \ldots, A_{r-1}) + H(A_1, A_2, \ldots, A_{r-1})$$

so we see that

$$H(A_1, A_2, \ldots, A_r) = H(A_r|A_{r-1}) + H(A_{r-1}|A_{r-2}) + \ldots + H(A_2|A_1) + H(A_1) = (r-1)H(A_2|A_1) + H(A_1).$$

Hence we see that the expected code length per letter tends to $H(A_2|A_1)/\log_2 D$ as r increases to infinity.

(In the situation described here, the successive letters A_j form a Markov chain with the probability distribution $p(a)$ as the invariant measure.)

5.2 Compression

The maximum value for the entropy $H(A_1)$ arises when all the K letters are equally likely and is $\log_2 K$. In this case, the optimal code satisfies

$$\log_D K = \frac{H(A_1)}{\log_2 D} \leqslant \mathbb{E}|c(A_1)| < 1 + \frac{H(A_1)}{\log_2 D} = \log_D(DK) \ .$$

There are K possibilities for letters from $\mathcal{A}$. Each letter in the code word $c(a)$ has D possibilities so the number of code words of length L is D^L and this is K when $L = \log_D K$, which is approximately the expected code length. Hence, in this case we do not get any compression of the message.

However, if the distribution of the letters is not uniform, then the entropy $H(A_1)$ is strictly less than $\log_2 K$ and so the optimal code does compress the message.

If we use block codes and the letters in our messages are not independent then there is much greater scope for compression. For then we know that

$$\frac{H(A_1, A_2, \ldots, A_r)}{r} < H(A_1)$$

so the expected code length per letter will be strictly smaller than

$$\frac{1}{r} + \frac{H(A_1, A_2, \ldots, A_r)}{r \log_2 D} < \frac{1}{r} + \frac{H(A_1)}{\log_2 D} \ .$$

The entropy $\dfrac{H(A_1, A_2, \ldots, A_r)}{r \log_2 D}$ measures the amount of information per letter in a block of length r. This gives us an upper limit on the amount by which we can compress the message (on average) without loosing information.

6: NOISY CHANNELS

So far we have assumed that each coded message is transmitted without error through the channel to the receiver where it can be decoded. In many practical applications, there is a small but appreciable probability of error, so a letter may be altered in the channel. We want to devise codes that detect, or even correct, such errors.

6.1 Memoryless Channels

A coded message $b_1 b_2 b_3 \dots b_N \in \mathcal{B}^*$ is sent through a channel C where each letter b_n may be altered to another. We will assume that any changes to one letter are independent of the other letters and what happens to them. In this case we say the channel is *memoryless.* So, if we write B_n for the random variable that gives the nth letter of the message and B'_n for the random variable giving the nth letter received, then

$$\mathbb{P}(b'_1 b'_2 b'_3 \dots b'_N \text{ received } | b_1 b_2 b_3 \dots b_N \text{ sent }) = \prod_{n=1}^{N} \mathbb{P}(B'_n = b'_n | B_n = b_n) \ .$$

This gives a *transition matrix*

$$\mathbb{P}(B'_n = i | B_n = j) \qquad \text{for } i, j \in \mathcal{B}$$

describing how the nth letter is altered. We expect the probability of error to be small, so the probability $\mathbb{P}(B'_n = i | B_n = i)$ should be close to 1.

We will also assume that the chances of a particular alteration does not depend on the index n, so the channel is *time independent.* Then the transition matrices do not change with n.

Example:
Let $\mathcal{B} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and the transition matrix be

$$\mathbb{P}(j \text{ received } | i \text{ sent }) = \begin{cases} \frac{3}{4} & \text{when } j = i; \\ \frac{1}{4} & \text{when } j = i + 1; \\ 0 & \text{otherwise.} \end{cases}$$

If I send a word $b_1 b_2 b_3 \dots b_N$, the probability that it is received without error is only $(\frac{3}{4})^N$.

I can improve the chances of receiving the message without error by sending each letter 3 times in succession. The probability that at least 2 of the 3 are received without error is then

$$\left(\tfrac{3}{4}\right)^3 + 3\left(\tfrac{1}{4}\right)\left(\tfrac{3}{4}\right)^2 = \tfrac{27}{32} = 0.84375$$

which is larger than $\frac{3}{4}$. So we have increased the probability of decoding the message without error to $\left(\frac{27}{32}\right)^N$ albeit at the price of sending a message three times as long.

We can do much better than this. For suppose that we first recode our message using the shorter alphabet $\mathcal{E} = \{0, 2, 4, 6, 8\}$. If a letter $i \in \mathcal{E}$ is sent, then either i or $i + 1$ is received. In either case we can tell **with certainty** that i was sent. So we can decode the message perfectly.

6.2 The Binary Symmetric Channel

For most of this course we will use binary codes for simplicity. When $\mathcal{B} = \{0, 1\}$, the *binary symmetric channel* (BSC) is a time independent, memoryless channel with transition matrix

$$\begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix} \ .$$

Here p is the probability of error in a single bit. Usually it is small. Note that if $p = \frac{1}{2}$ then no information is transmitted, everything is noise. When $p = 0$ there is no noise and no errors to be corrected.

Exercise:
In a binary symmetric channel we usually take the probability p of error to be less than 1/2. Why do we not consider $1 \geqslant p \geqslant 1/2$?

Example:
Punched computer tapes traditionally had 8 spaces per row and used holes to indicate the 1 bit. Of these bits the first 7 $(x_1, x_2, \ldots, x_7)$ carried information while the last x_8 was a check digit satisfying:

$$x_1 + x_2 + \ldots x_7 + x_8 \equiv 0 \pmod{2} \ . \qquad *$$

We can model the chances of errors in the punched tape using a binary symmetric channel with error probability p. Then the probability of receiving the information $x_1 x_2 \ldots x_7$ without error is $(1-p)^7$. If there is a one error in the 8 bits $x_1 x_2 \ldots x_7 x_8$ then $(*)$ will not hold for the received message and this will show that there was an error. In this case the information could be sent again.

Note that, when 2 errors occur, $(*)$ remains true and we do not know that there has been an error.

For instance, if $p = 0.1$, then the probability of 0 errors in $x_1 x_2 \ldots x_7$ is 0.48, the probability of 1 error that is detected by the check digit is 0.33.

Exercise:
Show that the probability of detecting errors at all using $(*)$ is $\frac{1}{2}(1 - (1-2p)^8)$.

6.3 Check Digits

Check digits are used very widely to detect errors.

Example:
University Candidate Numbers are of the form

$$1234A, \ 1235B, \ 1236C, \ 1237D, \ \ldots \ .$$

The first 4 digits identify the candidate, while the final letter is a check digit. If a candidate writes one letter incorrectly, then he or she will produce an invalid candidate number. The desk number can then be used to correct the error.

Exercise:
Books are identified by the ISBN-10 code. For example:

ISBN 0-521-404568

This consists of 9 decimal digits x_1-$x_2x_3x_4$-$x_5x_6x_7x_8x_9$ which identify the publisher, author and title. The final digit $x_{10} \in \{0, 1, 2, \ldots, 9, X\}$ is a check digit which satisfies

$$10x_1 + 9x_2 + 8x_3 + \ldots + 2x_9 + x_{10} \equiv 0 \pmod{11} \ .$$

Check the ISBN number above is valid. Show that altering any single digit or transposing any two adjacent digits in an ISBN-10 code will always result in an invalid code. Hence such errors can be detected.

How do check digits work in ISBN-13? Do they detect all single digit errors?

6.4 The Hamming Code

In the earlier lectures we saw that, when there was redundant information in a message, we could compress it. Alternatively, we can expand a code to add redundant information. This gives us the possibility of detecting errors or even correcting them.

Hamming produced a binary code that can correct a single error. This code takes a binary word of length 4 and codes it as a binary word of length 7. When we receive a word we look for the closest possible code word. Provided that there has been no more than one error in the 7 bits, we can always find that code word correctly and hence decode it.

The integers modulo 2 form a field $\mathbb{F}_2$ and we can use these to represent binary bits. So a binary string of length N is represented by a vector in the N-dimensional vector space $\mathbb{F}_2^N$ over $\mathbb{F}_2$. Hamming's code is given by a linear map $C : \mathbb{F}_2^4 \to \mathbb{F}_2^7$. This map is given by the matrix:

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

So a word $\boldsymbol{x} \in \mathbb{F}_2^4$ is coded as

$$C\boldsymbol{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_2 + x_3 + x_4 \\ x_1 + x_3 + x_4 \\ x_1 + x_2 + x_4 \end{pmatrix} .$$

The first 4 bits carry the information and the remaining 3 are check digits.

We can check that each column $\boldsymbol{y}$ of the matrix C satisfies

$$\begin{aligned} y_1 + y_3 + y_5 + y_7 &= 0 \\ y_2 + y_3 + y_6 + y_7 &= 0 \\ y_4 + y_5 + y_6 + y_7 &= 0 \ . \end{aligned} \tag{$\dagger$}$$

Hence each code word $\boldsymbol{y} = C\boldsymbol{x}$ must also satisfy these conditions (†). The converse is also true. Suppose that $\boldsymbol{y}$ satisfies (†). Then the equations determine y_5, y_6, y_7 in terms of y_1, y_2, y_3, y_4. So we see that

$$\boldsymbol{y} = C\boldsymbol{x} \qquad \text{where} \qquad \boldsymbol{x} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \in \mathbb{F}_2^4 \ .$$

Hence the equations (†) determine precisely when $\boldsymbol{y}$ is a code word or not.

We can rewrite the linear equations (†) in matrix terms as

$$S\boldsymbol{y} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \boldsymbol{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} .$$

This matrix S is called the *syndrome matrix.* We have seen that $SC\boldsymbol{x} = \boldsymbol{0}$ for every string $\boldsymbol{x} \in \mathbb{F}_2^4$. Indeed, $S \circ C = 0$ and the image $C(\mathbb{F}_2^4)$ is equal to the kernel $\ker S$.

Suppose that the string $\boldsymbol{x} \in \mathbb{F}_2^4$ is encoded as $C\boldsymbol{x}$ but an error occurs in the jth bit and nowhere else. Then we receive

$$\boldsymbol{y} = C\boldsymbol{x} + \boldsymbol{e}_j$$

where $\boldsymbol{e}_j$ is the vector with 1 in the jth place and 0s elsewhere. We know that $C\boldsymbol{x}$ satisfies (†) but $\boldsymbol{e}_j$ does not. Hence

$$S\boldsymbol{y} = SC\boldsymbol{x} + S\boldsymbol{e}_j \ ,$$

which is the jth column of the syndrome matrix S. Note that this jth column is simply the bits of j expanded in binary and reversed. For example

$$S\boldsymbol{e}_6 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \qquad \text{corresponds to} \qquad 110_2 = 6 \ .$$

In this way we can tell where the error occurred. Since the error is a change from 0 to 1 or 1 to 0, we can then correct it.

If I receive a message $\boldsymbol{y}$ in which two bits have errors, say $\boldsymbol{y} = C\boldsymbol{x} + \boldsymbol{e_i} + \boldsymbol{e}_j$, then the syndrome is

$$S\boldsymbol{y} = S\boldsymbol{e}_i + S\boldsymbol{e}_j \ .$$

Since i and j are different, the syndromes $S\boldsymbol{e}_i$ and $S\boldsymbol{e}_j$ are also different. So their sum $S\boldsymbol{e}_i + S\boldsymbol{e}_j$ is not zero. Hence $S\boldsymbol{y}$ is non-zero and tells us that there have been errors. However, it does not tell us what those errors are. For example, if we receive the string 1000000, then the syndrome is 100. This may have arisen from the code word 0000000 with one error or the code word 1000011 with two errors.

To summarise, the Hamming code detects 2 errors and can correct 1 error.

We can measure the distance between two vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ in $\mathbb{F}_2^N$ by counting the number of places where they differ. This gives

$$d(\boldsymbol{y}, \boldsymbol{z}) = |\{j : y_j \neq z_j\}| \ ,$$

which is called the *Hamming distance*. So the Hamming distance is the number of bit errors required to change $\boldsymbol{y}$ to $\boldsymbol{z}$. It is simple to see that this is a metric on $\mathbb{F}_2^N$ and we will always use this metric.

Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be two different strings in $\mathbb{F}_2^4$, and $C\boldsymbol{x}, C\boldsymbol{x}'$ the corresponding code words. Their difference

$$C\boldsymbol{x}' - C\boldsymbol{x} = C(\boldsymbol{x}' - \boldsymbol{x})$$

is the sum of some of the columns of the matrix C. It is easy to check that each such sum has at least 3 non-zero bits. So $d(C\boldsymbol{x}', C\boldsymbol{x})$ must be at least 3. It can be 3, for example $d(1000011, 0001111) = 3$.

Suppose that we receive a string $\boldsymbol{y} \in \mathbb{F}_2^7$ and wish to decode it. If $\boldsymbol{y}$ is a code word $C\boldsymbol{x}$ then we decode it as $\boldsymbol{x}$. If there is one error in $\boldsymbol{y}$, we can use the syndrome to find and correct that error. So we find a string $\boldsymbol{x} \in \mathbb{F}_2^4$ with $d(\boldsymbol{y}, C\boldsymbol{x}) = 1$. If there are two errors, it is natural to look for the string $\boldsymbol{x} \in \mathbb{F}_2^4$ for which $C\boldsymbol{x}$ is closest to $\boldsymbol{y}$, that is

$$d(\boldsymbol{y}, C\boldsymbol{x})$$

is minimal. We would then guess that $\boldsymbol{y}$ arose from $\boldsymbol{x}$ with $d(\boldsymbol{y}, C\boldsymbol{x})$ errors. The guess may not be unique. For example, if we receive 0000001, this is at distance 2 from both $1000011 = C(1000)$ and $0100101 = C(0100)$.

7: ERROR CORRECTING CODES

In this lecture we will consider only codes $c : \mathcal{A} \to \mathcal{B}^N$ of constant length N. We will use the discrete metric on $\mathcal{B}$:

$$d(b,v) = \begin{cases} 0 & \text{when } b = v; \\ 1 & \text{when } b \neq v\ . \end{cases}$$

The *Hamming distance* on $\mathcal{B}^N$ is then

$$d(\boldsymbol{b},\boldsymbol{v}) = \sum_{j=1}^{N} d(b_j, v_j)\ .$$

It is clear that this is a metric on $\mathcal{B}^N$. Note that the Hamming distance $d(\boldsymbol{c},\boldsymbol{v})$ counts the number of co-ordinates where $\boldsymbol{b}$ and $\boldsymbol{v}$ differ. When a word $\boldsymbol{b} \in \mathcal{B}^N$ is transmitted through a noisy channel, we receive an altered word $\boldsymbol{v}$. The Hamming distance $d(\boldsymbol{b},\boldsymbol{v})$ counts the number of letters that have been altered.

7.1 Decoding with Errors

Suppose that a word $\boldsymbol{b} \in \mathcal{B}^N$ is sent through a noisy channel and received as $\boldsymbol{v} \in \mathcal{B}^N$. We then wish to decode this. Because of the errors introduced by the channel, the word we receive may not be a code word at all and certainly not the code word $\boldsymbol{b}$ that was sent. How should we guess $\boldsymbol{b}$ when we know $\boldsymbol{v}$?

We will consider, briefly, three possible decoding rules. Suppose that $\boldsymbol{v} \in \mathcal{B}^N$ has been received.

Ideal Observer
Decode $\boldsymbol{v}$ as the word $\boldsymbol{b}$ with $\mathbb{P}(\boldsymbol{b} \text{ sent } | \ \boldsymbol{v} \text{ received })$ maximal. This is a sensible choice if we know enough to find the maximum. In most cases we do not.

Maximum Likelihood
Decode $\boldsymbol{v}$ as the word $\boldsymbol{b}$ with $\mathbb{P}(\boldsymbol{v} \text{ received } | \ \boldsymbol{b} \text{ sent })$ maximal. This is usually easier to find.

Minimum Distance
Decode $\boldsymbol{v}$ as the word $\boldsymbol{b}$ with $d(\boldsymbol{b},\boldsymbol{v})$ minimal.

Proposition 7.1
If all the code words are equally likely, then the ideal observer and maximum likelihood rules give the same result.

Proof:
We know that

$$\mathbb{P}(\boldsymbol{v} \text{ received } | \ \boldsymbol{b} \text{ sent }) = \frac{\mathbb{P}(\boldsymbol{v} \text{ received and } \boldsymbol{b} \text{ sent })}{\mathbb{P}(\boldsymbol{b} \text{ sent })} = \mathbb{P}(\boldsymbol{b} \text{ sent } | \ \boldsymbol{v} \text{ received })\frac{\mathbb{P}(\boldsymbol{v} \text{ received })}{\mathbb{P}(\boldsymbol{b} \text{ sent })}\ .$$

So, if all the code words are equally likely, then the ideal observer rule and the maximum likelihood rule give the same result. □

Proposition 7.2
If letters are transmitted through a binary symmetric channel with error probability $p < \frac{1}{2}$, then the maximum likelihood and minimum distance rules give the same result.

Proof:

$$\mathbb{P}(\boldsymbol{v} \text{ received } | \ \boldsymbol{b} \text{ sent }) = \prod_{j=1}^{N} \mathbb{P}(v_j \text{ received } |b_j \text{ sent }) = p^d(1-p)^{N-d} = (1-p)^N \left(\frac{p}{1-p}\right)^d$$

where d is the number of terms where $v_j \neq b_j$. Hence d is the Hamming distance $d(\boldsymbol{v}, \boldsymbol{b})$. Since $0 \leqslant p/(1-p) < 1$ we see that $\mathbb{P}(\boldsymbol{v} \text{ received } | \ \boldsymbol{b} \text{ sent })$ is maximal when $d(\boldsymbol{b}, \boldsymbol{v})$ is minimal. □

We will generally use the minimum distance decoding rule.

7.2 Error Detecting and Error Correcting

Throughout this section we will use binary codes $c : \mathcal{A} \to \{0,1\}^N$. This is for simplicity since the results can easily be adapted to more general alphabets. Recall that the *code words* are the words $c(a)$ for $a \in \mathcal{A}$. We will write K for the number of these $K = |\mathcal{A}|$. If a word $\boldsymbol{b} = c(a)$ is sent, then we receive a word $\boldsymbol{v} \in \{0,1\}^N$. We decode this by finding the code word $c(a')$ closest to $\boldsymbol{v}$, so $d(c(a'), \boldsymbol{v})$ is minimal.

The ball of radius r centred on $\boldsymbol{b} \in \{0,1\}^N$ is

$$B(\boldsymbol{b}, r) = \{\boldsymbol{v} \in \{0,1\}^N : d(\boldsymbol{b}, \boldsymbol{v}) < r\} .$$

Its *volume* is the number of points that it contains:

$$|B(\boldsymbol{b}, r)| = \sum_{0 \leqslant k < r} \binom{N}{k} .$$

This is clearly independent of the centre $\boldsymbol{b}$ and we will denote it by $V(N, r)$. It is reasonably simple to estimate the size of $V(N, r)$ and we will do so later.

The code $c : \mathcal{A} \to \{0,1\}^N$ is *e-error detecting* if, whenever no more than e letters in a code word $c(a)$ are altered, the received word is not a different code word $c(a')$. This means that we can tell if there have been errors, provided that there are at most e of them.

The code $c : \mathcal{A} \to \{0,1\}^N$ is *e-error correcting* if, whenever no more than e letters in a code word $c(a)$ are altered, the received word is still decoded as a using the minimum distance decoding rule.

The *minimum distance* for a code $c : \mathcal{A} \to \{0,1\}^N$ is

$$\delta = \min \{d(c(a), c(a')) : a, a' \in \mathcal{A} \ \text{ with } a \neq a'\} .$$

For this we immediately have:

Proposition 7.3 Minimum distance for a code
Let the code $c : \mathcal{A} \to \{0,1\}^N$ have minimum distance $\delta > 0$.

(a) c is $(\delta - 1)$-error detecting and not δ-error detecting.

(b) c is $\lfloor \frac{1}{2}(\delta - 1) \rfloor$-error correcting but not $\lfloor \frac{1}{2}(\delta + 1) \rfloor$-error correcting

Note that $\lfloor \frac{1}{2}(\delta+1) \rfloor = \lfloor \frac{1}{2}(\delta-1) \rfloor + 1$.

Proof:

(a) Clearly no code word other than $c(a)$ can lie within a distance $\delta - 1$ of $c(a)$. However, there are two letters $a_1, a_2 \in \mathcal{A}$ with $d(c(a_1), c(a_2)) = \delta$, so c is not δ-error detecting.

(b) If $a' \neq a$, and $d(c(a), \boldsymbol{v}) \leqslant \lfloor \frac{1}{2}(\delta-1) \rfloor$, then the triangle inequality gives

$$d(c(a'), \boldsymbol{v}) \geqslant d(c(a'), c(a)) - d(c(a), \boldsymbol{v}) \geqslant \delta - \lfloor \tfrac{1}{2}(\delta-1) \rfloor > \lfloor \tfrac{1}{2}(\delta-1) \rfloor \ .$$

So c is $\lfloor \frac{1}{2}(\delta-1) \rfloor$-error detecting. However, there are two letters $a_1, a_2 \in \mathcal{A}$ with $d(c(a_1), c(a_2)) = \delta$, so $c(a_1)$ and $c(a_2)$ differ in exactly δ places. Choose $\lfloor \frac{1}{2}(\delta+1) \rfloor$ of these places and change $c(a_1)$ in these to get a word $\boldsymbol{v}$ with

$$d(\boldsymbol{v}, c(a_1)) = \lfloor \tfrac{1}{2}(\delta+1) \rfloor \qquad \text{and} \qquad d(\boldsymbol{v}, c(a_2)) = \delta - \lfloor \tfrac{1}{2}(\delta+1) \rfloor \leqslant \lfloor \tfrac{1}{2}(\delta+1) \rfloor \ .$$

So c can not be $\lfloor \frac{1}{2}(\delta+1) \rfloor$-error correcting. □

Example:
For the repetition code, where each letter is repeated m-times, the minimum distance is m, so this code is $(m-1)$-error detecting and $\lfloor \frac{1}{2}(m-1) \rfloor$-error correcting.

For a code $\mathcal{A} \to \{0,1\}^N$ where the Nth bit is a check digit, the minimum distance is 2, so this code is 1-error detecting and 0-error correcting.

For the Hamming code $h : \{0,1\}^4 \to \{0,1\}^7$, the minimum distance is 3, so the Hamming code is 2-error detecting and 1-error correcting.

7.3 Covering Estimates

Observe that the code $c : \mathcal{A} \to \{0,1\}^N$ is e-error correcting precisely when the balls $B(c(a), e+1)$ for $a \in \mathcal{A}$ are disjoint. We can use this to estimate the number of code words for such a code.

Proposition 7.4 Hamming's bound
If $c : \mathcal{A} \to \{0,1\}^N$ is an e-error correcting code, then the number of code words $K = |\mathcal{A}|$ must satisfy

$$K \leqslant \frac{2^N}{V(N, e+1)} \ .$$

Proof:
First note that

$$B(\boldsymbol{c}, e+1) = \{\boldsymbol{v} \in \{0,1\}^N : d(\boldsymbol{c}, \boldsymbol{v}) < e+1\} = \{\boldsymbol{v} \in \{0,1\}^N : d(\boldsymbol{c}, \boldsymbol{v}) \leqslant e\}$$

so the open ball $B(\boldsymbol{c}, e+1)$ is the closed ball of radius e.

When c is e-error correcting, the balls $B(c(a), e+1)$ for $a \in \mathcal{A}$ must be disjoint. Hence their total volume $KV(N, e+1)$ is at most equal to the volume of all of $\{0,1\}^N$, which is 2^N. □

A code is said to be *perfect* if it is e-error correcting for some e and the balls $B(c(a), e+1)$ cover all of $\{0,1\}^N$. This is the case when there is equality in Hamming's bound. For example, Hamming's code is 1-error correcting, $K = 2^4$, $N = 7$ and

$$V(7,2) = \binom{7}{0} + \binom{7}{1} = 8 \ .$$

So $K = 2^4 = 2^7/8 = 2^N/V(7,2)$ and the code is perfect.

We can also use a similar idea to give a lower bound.

Proposition 7.5 Gilbert – Shannon - Varshamov bound
There is an e-error detecting code $c : \mathcal{A} \to \{0,1\}^N$ with the number of code words $K = |\mathcal{A}|$ satisfying

$$K \geqslant \frac{2^N}{V(N, e+1)} \ .$$

Proof:
For an e-error detecting code, no code word other than $c(a)$ can lie within the ball $B(c(a), e+1)$. Choose a maximal set $\mathcal{C}$ of code words satisfying this condition and let $K = |\mathcal{C}|$. If there were any word $\boldsymbol{v}$ not in the union

$$\bigcup_{\boldsymbol{c} \in \mathcal{C}} B(\boldsymbol{c}, e+1)$$

then we could add it to $\mathcal{C}$, contradicting maximality. Therefore,

$$\bigcup_{\boldsymbol{c} \in \mathcal{C}} B(\boldsymbol{c}, e+1) = \{0,1\}^N \ .$$

The volume of each ball is $V(N, e+1)$, so the volume of their union is no more than $KV(N, e+1)$. Hence $KV(N, e+1) \geqslant 2^N$. □

Corollary 7.6
Let $c : \mathcal{A} \to \{0,1\}^N$ be a code with minimum distance δ. Then

$$K = |\mathcal{A}| \leqslant \frac{2^N}{V(N, \lfloor \frac{1}{2}(\delta+1) \rfloor)} \ .$$

Moreover, there is a code with minimum distance δ and

$$K = |\mathcal{A}| \geqslant \frac{2^N}{V(N, \delta)} \ .$$

□

7.4 Asymptotics for $V(N,r)$

We wish to use the Hamming and Gilbert – Shannon – Varshamov bounds of the last section to give asymptotic estimates on how large error correcting codes are. To do this we need to determine the asymptotic behaviour of the volumes $V(N,r)$. It will be useful to describe this in terms of the entropy of a Bernoulli random variable that takes two values with probability q and $1-q$. We will denote this by

$$h(q) = -q \log_2 q - (1-q) \log_2 (1-q) \ .$$

Lemma 7.7 Asymptotics of binomial coefficients
For natural numbers $0 \leqslant r \leqslant N$ we have

$$\frac{1}{N+1} 2^{Nh(q)} \leqslant \binom{N}{r} \leqslant 2^{Nh(q)}$$

where $q = r/N$.

Proof:

Let X be a Bernoulli $B(N,q)$ random variable, so

$$\mathbb{P}(X=k) = \binom{N}{k} q^k (1-q)^{N-k} \ .$$

(The entropy of X is $H(X) = Nh(q)$.) It is simple to check that this probability is maximal when $k = r = qN$. Therefore,

$$\mathbb{P}(X=r) \ \leqslant \ \sum_{k=0}^{N} \mathbb{P}(X=k) \ = \ 1 \ \leqslant \ (N+1)\mathbb{P}(X=r) \ .$$

This means that

$$\frac{1}{N+1} \ \leqslant \ \mathbb{P}(X=r) = \binom{N}{r} q^r (1-q)^{N-r} = \binom{N}{r} 2^{-Nh(q)} \ \leqslant 1 \ .$$

The result follows when we observe that

$$2^{-Nh(q)} = q^{Nq}(1-q)^{N(1-q)} = q^r(1-q)^{N-r} \ .$$

□

Exercise:
Use Stirling's formula $\left[n! \sim \sqrt{2\pi n}\dfrac{n^n}{e^n} \right]$ to describe the behaviour of $\binom{N}{r}$ in terms of the entropy $h(r/N)$.

Proposition 7.8 Asymptotics of $V(N,r)$
For a natural number N and $0 < r < \frac{1}{2}N$

$$V(N,r) \ \leqslant \ 2^{Nh(q)}$$

for $q = r/N$ and

$$\frac{1}{N+1} 2^{Nh(q')} \ \leqslant \ V(N,r)$$

for $q' = (\lceil r \rceil - 1)/N$.

Proof:

$V(N,r)$ is the number of points in an open ball of radius r in $\{0,1\}^N$, so

$$V(N,r) = \sum_{0 \leqslant k < r} \binom{N}{k} \ .$$

Since $q = r/N \leqslant \frac{1}{2}$, we have $q^k(1-q)^{N-k} \leqslant q^r(1-q)^{N-r}$ for $0 \leqslant k < r$. So

$$\begin{aligned} 1 = \sum_{0 \leqslant k \leqslant N} \binom{N}{k} q^k (1-q)^{N-k} &\geqslant \sum_{0 \leqslant k < r} \binom{N}{k} q^k (1-q)^{N-k} \\ &\geqslant \left(\sum_{0 \leqslant k < r} \binom{N}{k} \right) q^r (1-q)^{N-r} = V(N,r) q^r (1-q)^{N-r} \ . \end{aligned}$$

This shows that $V(N,r) \ \leqslant \ 2^{Nh(q)}$.

For the other inequality, let $k = \lceil r \rceil - 1$ so k is the largest integer strictly smaller than r. Then

$$V(N,r) \geqslant \binom{N}{k} \geqslant \frac{1}{N+1} 2^{Nh(q')}$$

because of Lemma 7.7 . □

Consider a code $c : \mathcal{A} \to \{0,1\}^N$. If A is a a random variable taking values in the alphabet $\mathcal{A}$ then information is transmitted through the channel at a rate

$$\frac{H(A)}{N} \ .$$

This is largest when A is equally distributed over the K possible values. Then it is

$$\frac{\log_2 K}{N} \ .$$

Choose q with $0 < q < \frac{1}{2}$. Hamming's bound (Proposition 7.4) shows that

$$\frac{\log_2 K}{N} \leqslant 1 - \frac{\log_2 V(N, qN)}{N}$$

for $(qN-1)$-error correcting codes. The last proposition now shows that the right side tends to $1-h(q)$ as $N \to \infty$.

Similarly, the Gilbert – Shannon – Varshamov bound Proposition 7.5 shows that there are $(qN-1)$-error detecting codes with information rate

$$\frac{\log_2 K}{N} \geqslant 1 - \frac{\log_2 V(N, qN)}{N}$$

and the right side tends to $1 - h(q)$ as $N \to \infty$.

8: INFORMATION CAPACITY

8.1 Mutual Information

We wish to determine how much information can be passed successfully through a noisy channel for each bit that is transmitted. Consider a time-independent, memoryless channel. If a random letter $B \in \mathcal{B}$ is sent through the channel, then a new random letter B' is received. Because the channel is noisy, B' may differ from B.

The average amount of information given by B is the entropy $H(B)$, while the average amount of information received is $H(B')$. The information received is partly due to B and partly due to the noise. The conditional entropy $H(B'|B)$ is the amount of information in B' conditional on the value of B, so it is the information due to the noise in the channel. The remaining information

$$H(B') - H(B'|B)$$

is the part that is due to the letter B that was sent. It is the amount of uncertainty about B' that is removed by knowing B. We therefore define the *mutual information of B and B'* to be

$$I(B', B) = H(B') - H(B'|B) \ .$$

Proposition 8.1 Mutual Information
The mutual information satisfies:

$$I(B', B) = H(B') + H(B) - H(B', B) = I(B, B') \ .$$

Furthermore,

(a) $I(B', B) \geqslant 0$ with equality if and only if B' and B are independent.

(b) $I(B', B) \leqslant H(B')$ with equality if and only if B' is a function of B.

(c) $I(B', B) \leqslant H(B)$ with equality if and only if B is a function of B'.

Proof:
Recall from Lecture 2 that the conditional entropy satisfies $H(B'|B) = H(B', B) - H(B)$, so

$$I(B', B) = H(B') - H(B'|B) = H(B') - H(B', B) + H(B)$$

and the left side is clearly symmetric.

(a) Corollary 2.2 shows that $I(B', B) \geqslant 0$ with equality if and only if B and B' are independent.

(b) We always have $H(B'|B) \geqslant 0$ with equality if and only if B' is a function of B.

(c) Since $I(B', B) = I(B, B')$ part (c) follows from (b). □

In our case, the distribution $\mathbb{P}(B = b)$ is known as are the transition probabilities $\mathbb{P}(B' = b'|B = b)$. From these we can calculate both

$$\mathbb{P}(B' = b', B = b) = \mathbb{P}(B' = b'|B = b)\mathbb{P}(B = b)$$

and

$$\mathbb{P}(B' = b') = \sum_{b \in \mathcal{B}} \mathbb{P}(B' = b'|B = b)\mathbb{P}(B = b) \ .$$

Thence we can find the entropies $H(B), H(B'), H(B', B)$ and the mutual information. If the channel has no noise, then $B' = B$ and the proposition shows that the mutual information is just the entropy $H(B)$.

The *information capacity* of the channel is the supremum of the mutual information $I(B', B)$ over all probability distributions for B. The probability distribution of B is given by a point $\boldsymbol{p} = (p(a_1), p(a_2), \ldots, p(A_K))$ in the compact set $\{\boldsymbol{p} \in [0, 1]^K : \sum p_k = 1\}$. The mutual information is a continuous function of $\boldsymbol{p}$, so we know that the supremum is attained for some distribution. This information capacity depends only on the transition probabilities $\mathbb{P}(B'|B)$.

Proposition 8.2 Information capacity of a BSC
A binary symmetric channel with probability p of error has information capacity $1 - h(p)$.

Proof:
Suppose that the letter B has the distribution $\mathbb{P}(B = 1) = t \ , \ \ \mathbb{P}(B = 0) = 1 - t$. Then the entropy of B is

$$H(B) = H(1 - t, t) = -t \log_2 t - (1 - t) \log_2(1 - t) = h(t) \ .$$

The conditional entropy is

$$\begin{aligned} H(B'|B) &= \mathbb{P}(B = 1)H(B'|B = 1) + \mathbb{P}(B = 0)H(B'|B = 0) \\ &= tH(1 - p, p) + (1 - t)H(p, 1 - p) = h(p) \ . \end{aligned}$$

The distribution of B' is

$$\mathbb{P}(B' = 1) = t(1 - p) + (1 - t)p = t + p - 2tp \ , \ \ \mathbb{P}(B' = 0) = tp + (1 - t)(1 - p) = 1 - t - p + 2tp$$

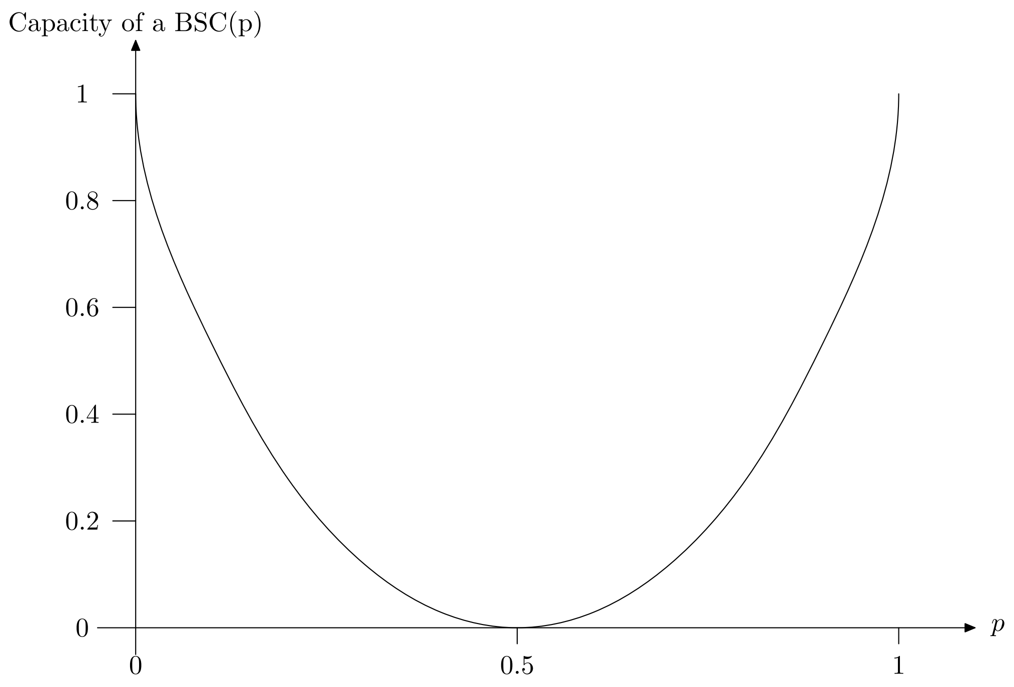
so its entropy is

$$H(B') = h(t + p - 2tp) \ .$$

Therefore the mutual information is

$$I(B', B) = H(B') - H(B'|B) = h(t + p - 2tp) - h(p) \ .$$

We can choose any distribution for B and so take any t between 0 and 1. The maximum value for $h(t + p - 2tp)$ is 1 which is attained when $t = \frac{1}{2}$ and $t + p - 2tp = \frac{1}{2}$. So the information capacity of the binary symmetric channel is $1 - h(p)$. $\square$



Note that, when $p = 0$ or 1, the channel transmits perfectly and the information capacity is 1. When $p = \frac{1}{2}$, no information is transmitted and the capacity is 0.

We often wish to consider using the same channel repeatedly, for example, when we wish to transmit a word in $\{0,1\}^N$ by sending the N bits one at a time through a binary symmetric channel. We can also think of this as having N copies of the channel in parallel and sending the jth bit through the jth channel. It is simple to compute the capacity in such a situation.

Proposition 8.3 The capacity of parallel channels
Let Q be a channel that transmits letters from the alphabet $\mathcal{B}$ and has capacity $\mathrm{Cap}(Q)$. *Form a new channel Q^N that transmits an N-tuple $(b_1, b_2, \dots, b_N) \in \mathcal{B}^N$ one letter at a time through the channel Q with each use being independent of the others. Then the capacity of Q^N is $N\mathrm{Cap}(Q)$.*

Proof:
Let $\boldsymbol{B} = (B_1, B_2, \dots, B_N)$ be a random variable taking values in the product alphabet $\mathcal{B}^N$ and let $\boldsymbol{B}' = (B_1', B_2', \dots, B_N')$ be the random variable we receive after sending $\boldsymbol{B}$ through the channel Q^N.

For each vector $\boldsymbol{b} = (b_1, b_2, \dots, b_N)$ we know that

$$H(\boldsymbol{B}'|\boldsymbol{B} = \boldsymbol{b}) = \sum H(B_j'|B_j = b_j)$$

because, once we condition on the $\boldsymbol{B}$, each B_j' is independent of all the other letters received. Therefore,

$$H(\boldsymbol{B}'|\boldsymbol{B}) = \sum H(B_j'|B_j) \ .$$

Also,

$$H(\boldsymbol{B}') \leqslant \sum H(B_j')$$

with equality if and only if the (B_j') are independent.

Thus

$$I(\boldsymbol{B}', \boldsymbol{B}) = H(\boldsymbol{B}') - H(\boldsymbol{B}'|\boldsymbol{B}) \leqslant \sum_{j=1}^{N} H(B_j') - H(B_j'|B_j) = \sum_{j=1}^{N} I(B_j', B_j)$$

and there is equality if and only if the (B'_j) are independent. This shows that $\mathrm{Cap}(Q^N) \leqslant N\mathrm{Cap}(Q)$.

Now choose the distribution of B_j so that $\mathrm{Cap}(Q) = I(B'_j, B_j)$ and the (B_j) are independent. Then the (B'_j) are also independent and
$$I(\boldsymbol{B}', \boldsymbol{B}) = N\mathrm{Cap}(Q)$$
so the product channel Q^N has capacity $N\mathrm{Cap}(Q)$. □

Exercise:
Use a similar argument to show that the capacity of independent channels Q_j in parallel is $\sum \mathrm{Cap}(Q_j)$.

Suppose that we send a random variable X through a channel and receive Y with mutual information $I(X,Y)$. If we apply a function f to Y then it is intuitively clear that the information transmitted from X to $f(Y)$ is no larger than that transmitted from X to Y. The next Proposition proves this rigorously.

Proposition 8.4 Functions do not increase mutual information
Let X, Y be random variable and f a function defined on the values taken by Y, then
$$I(X, f(Y)) \leqslant I(X,Y)\ .$$
Similarly, $I(f(Y), X) \leqslant I(Y,X)$.

Proof:
We know from Corollary 2.2 that $H(A,B) \leqslant H(A) + H(B)$. Applying this to the random variables conditioned on the value of Z gives
$$H(X,Y|Z) \leqslant H(X|Z) + H(Y|Z)\ .$$
This is equivalent to
$$\begin{aligned} & H(X,Y,Z) - H(Z) \leqslant H(X,Z) - H(Z) + H(Y,Z) - H(Z) \\ \Leftrightarrow \quad & H(X) + H(Z) - H(X,Z) \leqslant H(X) + H(Y,Z) - H(X,Y,Z) \\ \Leftrightarrow \quad & I(X,Z) \leqslant I(X,(Y,Z)) \end{aligned}$$
If we set $Z = f(Y)$, then $H(Y,Z) = H(Y)$ and $H(X,Y,Z) = H(X,Y)$ so $I(X,(Y,Z)) = I(X,Y)$. Thus we have
$$I(X, f(Y)) \leqslant I(X,Y)$$
as required.

Since $I(X,Y)$ is symmetric (Proposition 8.1), we also have $I(X, f(Y)) \leqslant I(X,Y)$. □

Exercise:
Data Processing Inequality
Consider two independent channels in series. A random variable X is sent through channel 1 and received as Y. This is then sent through channel 2 and received as Z. Prove that $I(X,Z) \leqslant I(X,Y)$, so the further processing of the second channel can only reduce the mutual information. (The last proposition established this when the second channel has no noise.)

The independence of the channels means that, if we condition on the value of Y, then $(X|Y=y)$ and $(Z|Y=y)$ are independent. Deduce that
$$H(X,Z|Y) = H(X|Y) + H(Z|Y)\ .$$

By writing the conditional entropies as $H(A|B) = H(A,B) - H(B)$, show that
$$H(X,Y,Z) + H(Z) = H(X,Y) + H(Y,Z)\ .$$

Define $I(X,Z|Y)$ as $H(X|Y) + H(Z|Y) - H(X,Z|Y)$ and show that
$$I(X,Z|Y) = I(X,Y) - I(X,Z)\ .$$
Deduce from this the *data processing inequality*:
$$I(X,Z) \leqslant I(X,Y)\ .$$
When is there equality?

9: FANO'S INEQUALITY

9.1 A Model for Noisy Coding

We will, eventually, prove Shannon's Coding Theorem which shows that we can find codes that transmit messages through a noisy channel with small probability of error and at a rate close to the information capacity of the channel. These are very good codes. Unfortunately, Shannon's argument is not constructive and it has proved very difficult to construct such good codes. In the remainder of this section we will show that the rate at which a code transmits information can not exceed the information capacity of the channel without the probability of error being large.

The situation we face is as follows. We wish to transmit a message written in the alphabet $\mathcal{A}$ of size K. Let A be a random variable that takes values in $\mathcal{A}$. Then $H(A) \leqslant \log_2 K$ with equality when all K letters in $\mathcal{A}$ are equally likely. We will assume that this is the case.

We choose a constant length code $c : \mathcal{A} \to \mathcal{B}^N$ to encode the message, one letter at a time. This gives a random variable $\boldsymbol{B} = c(A)$ taking values in $\mathcal{B}^N$. Since c is assumed to be invertible, A determines $\boldsymbol{B}$ and *vice versa*. Therefore $H(A) = H(\boldsymbol{B})$.

Each bit of $c(a)$ is then passed through a channel where errors may arise. This results in a new string $c(a)' \in \mathcal{B}^N$ and we set $\boldsymbol{B}' = c(A)'$. The closest code word to $c(a)'$ will be denoted by $c(a')$ for some $a' \in \mathcal{A}$ and we then decode it as a'. The random variable A' results from decoding $\boldsymbol{B}'$. Since A' is a function of $\boldsymbol{B}'$, we certainly have $H(A') \leqslant H(\boldsymbol{B}')$.

The probability of error when the letter a is sent is

$$\mathbb{P}(\text{ error } \mid a \text{ sent })$$

and the total probability of error is

$$\mathbb{P}(\text{ error }) = \sum_{a \in \mathcal{A}} \mathbb{P}(\text{ error } \mid a \text{ sent })\mathbb{P}(A = a)\ .$$

We would like this to be small. Indeed, we would really like the probability to be small over all choices of $a \in \mathcal{A}$. So we want the *maximum error*

$$\widehat{e}(c) = \max \{\mathbb{P}(\text{ error } \mid a \text{ sent }) : a \in \mathcal{A}\}$$

to be small.

The *rate of transmission for c* is

$$\rho(c) = \frac{H(A)}{N} = \frac{\log_2 K}{N}\ .$$

We want this to be as large as possible. Our aim is to relate this to the information capacity of the channel.

		SENDER		CHANNEL	RECEIVER		
	$\mathcal{A}$	$\xrightarrow{c}$	$\mathcal{B}^N$	$\dashrightarrow$	$\mathcal{B}^N$	$\longrightarrow$	$\mathcal{A}$
Random Variables	A		$\boldsymbol{B}$	Cap	$\boldsymbol{B}'$		A'
Entropy	$H(A)$	$=$	$H(\boldsymbol{B})$		$H(\boldsymbol{B}')$	$\geqslant$	$H(A')$
		$\log_2 K$					

Note that the particular code $c : \mathcal{A} \to \mathcal{B}^N$ does not really matter. We can permute the code words $\{c(a) : a \in \mathcal{A}\}$ in any way we wish and not alter the rate of tranmission or the maximum error probability. So, the set of code words is really more important than the actual alphabet or code we choose. We sometimes refer to the set of code words $\{c(a) : a \in \mathcal{A}\}$ as the *code book.*

9.2 Fano's Inequality

Lemma 9.1
Let X be a random variable that takes values in a finite alphabet $\mathcal{A}$ of size K. Then, for any fixed letter $a \in \mathcal{A}$, we have

$$H(X) \leqslant p \log_2(K-1) + h(p)$$

where $p = \mathbb{P}(X \neq a)$.

Proof:
Recall that

$$H(X, I) = H(X|I) + H(I) \ .$$

Let I be the indicator random variable

$$I = \begin{cases} 1 & \text{when } X \neq a; \\ 0 & \text{when } X = a. \end{cases}$$

Then we see that:

$$H(X, I) = H(X)$$

because I is determined by the values of X.

$$\begin{aligned} H(X|I) &= \mathbb{P}(I=0)H(X|I=0) + \mathbb{P}(I=1)H(X|I=1) \\ &= (1-p)0 + pH(X|I=1) \\ &\leqslant p \log_2(K-1) \end{aligned}$$

because, if $I = 0$ then $X = a$ and so $H(X|I = 0) = 0$, while if $I = 1$, there are only $K-1$ possible values for X.

Therefore, we have

$$\begin{aligned} H(X) &= H(X|I) + H(I) \\ &\leqslant p \log_2(K-1) + h(p) \end{aligned}$$

because $H(I) = H(1-p, p) = h(p)$. □

Theorem 9.2 Fano's inequality
Let X, Y be two random variables that take values in a finite alphabet $\mathcal{A}$ of size K. Then

$$H(X|Y) \leqslant p \ \log_2(K-1) + h(p)$$

where $p = \mathbb{P}(X \neq Y)$.

We will apply this where X takes values in the alphabet $\mathcal{A}$ and Y is the result of passing the code word $c(X)$ through a channel and decoding it. The probability p is then the probability of error.

Proof:
Condition the inequality in the Lemma on the value of Y to obtain

$$H(X|Y = y) \leqslant \mathbb{P}(X \neq y|Y = y) \log_2(K-1) + H(I|Y = y)$$

where I is given by

$$I = \begin{cases} 1 & \text{when } X \neq Y; \\ 0 & \text{when } X = Y. \end{cases}$$

If we multiply this by $\mathbb{P}(Y = y)$ and sum over all values for y we obtain

$$H(X|Y) \leqslant \mathbb{P}(X \neq Y) \log_2(K-1) + H(I|Y) = p \log_2(K-1) + H(I|Y) \ .$$

Finally, recall that $H(I|Y) = H(I, Y) - H(Y) \leqslant H(I)$ by Corollary 2.2 . □

We can think of the two terms of Fano's inequality as $h(p) = H(I)$ measuring the information given by knowing whether there is an error and $p \log_2(K-1)$ measuring the information on what that error is, when it occurs.

We can use Fano's inequality to compare the rate of transmission of information to the capacity of the channel. As always we will use our model for a noisy channel. A random letter A is chosen from $\mathcal{A}$ and encoded as the $\mathcal{B}^N$-valued random variable $\boldsymbol{B} = c(A)$. This is transmitted, one letter at a time, through the noisy channel and received as the $\mathcal{B}^N$-valued random variable $\boldsymbol{B}'$ which is decoded to give A'. The rate of transmission is $\rho(c) = \log_2 K/N$.

Choose A so that each letter in $\mathcal{A}$ has probability $1/K$. Fano's inequality gives

$$H(A|A') \leqslant h(p) + p\log_2(K-1) < h(p) + p\log_2 K \ .$$

So we have

$$I(A', A) = H(A) - H(A|A') \geqslant \log_2 K - (h(p) + p\log_2 K) \ .$$

Now the code $c : \mathcal{A} \to \mathcal{B}^N$ is injective, so A is a function of $\boldsymbol{B}$. Therefore, Proposition 8.4 show that

$$I(A', A) \leqslant I(A', \boldsymbol{B}) \ .$$

Similarly, A' is a function of $\boldsymbol{B}'$ — the decoding function. So

$$I(A', \boldsymbol{B}) \leqslant I(\boldsymbol{B}', \boldsymbol{B}) \ .$$

Therefore, $I(A', A) \leqslant I(\boldsymbol{B}', \boldsymbol{B}) \leqslant N\mathrm{Cap}$ and hence

$$N\mathrm{Cap} \geqslant (1-p)\log_2 K - h(p) \ .$$

Dividing by N we obtain

$$\mathrm{Cap}(Q) + \frac{h(p)}{N} \geqslant (1-p)\rho(c) \ .$$

Theorem 9.3 Capacity bounds the rate of transmission of information
Let $c : \mathcal{A} \to \mathcal{B}^N$ where each letter of a code word is transmitted through a time-independent memoryless channel with capacity Cap. *Let p be the probability of decoding incorrectly. Then the rate of transmission satisfies*

$$\rho(c) \leqslant \frac{\log_2 K}{N} \leqslant \frac{\mathrm{Cap}}{1-p} + \frac{h(p)}{N(1-p)} \ .$$

□

As we let the probability of error decrease to 0, so $1 - p \nearrow 1$ and $h(p) \searrow 0$. Therefore

$$\frac{\mathrm{Cap}}{1-p} + \frac{h(p)}{N(1-p)} \searrow \mathrm{Cap} \ .$$

Hence, if we find codes c_j with probabilities of error p_j that decrease to 0, and rates of tranmission that tend to a limit ρ, then ρ must be at most the information capacity of the channel.

10: SHANNON'S NOISY CODING THEOREM

10.1 Introduction

In this section we will prove Shannon's Noisy Coding Theorem, which shows that we can find codes with rate of tranmission close to the capacity of a channel and small probability of error. We will do this only in the case of a binary symmetric channel. The arguments are simpler in this case but the principles apply more generally.

Theorem 10.1 Shannon's Noisy Coding Theorem
For $\varepsilon > 0$ and a binary symmetric channel with capacity Cap *there are codes*

$$c_N : \mathcal{A}_N \to \{0,1\}^N$$

for N sufficiently large with rate of tranmission

$$\rho(c_N) = \frac{\log_2 |\mathcal{A}_N|}{N} \geqslant \text{Cap} - \varepsilon$$

and maximum error

$$\widehat{e}(c_N) < \varepsilon .$$

Example:
We wish to send a message m from an alphabet $\mathcal{A}$ of size K through a binary symmetric channel with error probability $p = 0.1$. What rate of transmission can we achieve with small probability of error?

$\text{Cap} = 1 - h(0.1) = 0.53$, so Shannon's Theorem shows that we can achieve any rate of tranmission strictly less than this with arbitrarily small probability of error. For example, suppose that we want a rate of tranmission $\rho \geqslant 0.5$ and $\widehat{e} < 0.01$. Shannon's Theorem shows that there are codes $c_N : \mathcal{A}_N \to \{0,1\}^N$ that achieve this provided that N is large enough, say $N \geqslant N_o$.

Suppose that we know such a code c_N. How do we encode the message m? First divide m into blocks of length B where

$$B = \left\lfloor \frac{0.5N}{\log_2 K} \right\rfloor \qquad \text{so that} \qquad |\mathcal{A}^B| = K^B \leqslant 2^{0.5N} .$$

Then we can embed the blocks from $\mathcal{A}^B$ in the alphabet $\mathcal{A}_N$ and so encode the blocks. The rate of tranmission is

$$\frac{\log_2 |\mathcal{A}^B|}{N} \sim 0.5 .$$

Unfortunately, Shannon's Theorem tells us that there are such codes but does not tell us how to find them and it is very difficult to do so.

10.2 Chebyshev's Inequality

Theorem 10.2 Chebyshev's inequality
Let X be a real-valued, discrete random variable with mean $\mathbb{E}X$ and variance $\text{var}(X)$. Then

$$\mathbb{P}\left(|X - \mathbb{E}X| \geqslant t\right) \leqslant \frac{\text{var}(X)}{t^2} .$$

(The inequality is true whenever the variance of X exists, even if X is not discrete.)

Proof:

We have

$$\begin{aligned}\operatorname{var}(X) = \mathbb{E}|X - \mathbb{E}X|^2 &= \sum \mathbb{P}(X = x)|x - \mathbb{E}X|^2 \\ &\geqslant \sum \left\{\mathbb{P}(X = x)t^2 : |x - \mathbb{E}X| \geqslant t\right\} = t^2\mathbb{P}(|X - \mathbb{E}X| \geqslant t)\ .\end{aligned}$$

So we obtain the result. □

10.3 Proof of the Noisy Coding Theorem

To prove Shannon's Theorem (Theorem 10.1) we need to find code words in $\{0,1\}^N$. Choose these at random and independently of one another and of the channel. We will prove that, on average, this gives the inequalities we want. Then there must be at least some of the choices for code words that also give the inequalities.

Let the binary symmetric channel have error probability p with $0 \leqslant p < \frac{1}{2}$. Then $\text{Cap} = 1 - h(p)$. Set

$$K_N = \left\lfloor 2^{N(\text{Cap}-\varepsilon)} \right\rfloor \ \ll\ 2^N$$

and let $\mathcal{A}_N$ be an alphabet with K_N letters.

Choose a letter $a_o \in \mathcal{A}_N$ and then choose a random code word as $c_N(a_o)$ uniformly from the 2^N words in $\{0,1\}^N$. These choices must be independent for each different letter and also independent of the channel. When a fixed word $\boldsymbol{c}_o = c_N(a_o)$ is sent through the channel, it is corrupted and a new word $\boldsymbol{c}_o'$ is received. We want this received word to be close to $\boldsymbol{c}_o$.

The Hamming distance $d(\boldsymbol{c}, \boldsymbol{c}')$ is a Bernoulli $B(N,p)$ random variable, so it has mean Np and variance $Np(1-p)$. Therefore Chebyshev's inequality gives

$$\mathbb{P}(d(\boldsymbol{c}_o, \boldsymbol{c}_o') \geqslant r) \leqslant \frac{Np(1-p)}{(r - Np)^2}$$

for $r > Np$. Take $q > p$ and set $r = Nq$, then

$$\mathbb{P}(d(\boldsymbol{c}_o, \boldsymbol{c}_o') \geqslant r) \leqslant \frac{p(1-p)}{N(q-p)^2} \qquad (1).$$

Now consider another code word $\boldsymbol{c} = c_N(a)$ for a letter $a \neq a_o$. This is chosen uniformly and randomly from the 2^N words in $\{0,1\}^N$, so Proposition 7.8 shows that

$$\mathbb{P}(d(\boldsymbol{c}, \boldsymbol{c}_o') < r) = \mathbb{P}(\boldsymbol{c} \in B(\boldsymbol{c}_o', r)) = \frac{V(N,r)}{2^N} \leqslant \frac{2^{Nh(q)}}{2^N} = 2^{-N(1-h(q))}\ .$$

The probability that $d(\boldsymbol{c}, \boldsymbol{c}_o') < r$ for at least one of the $K_N - 1$ code words $\boldsymbol{c}$ not equal to $\boldsymbol{c}_o$ is therefore at most

$$K_N 2^{-N(1-h(q))} = 2^{N(\text{Cap}-\varepsilon-(1-h(q)))}\ .$$

Now $\text{Cap} = 1 - h(p)$, so this gives

$$\mathbb{P}(\text{ there exists } a \neq a_o \text{ with } d(c_N(a), \boldsymbol{c}_o') < r) \leqslant 2^{N(h(q)-h(p)-\varepsilon)} \qquad (2).$$

We will use the minimum distance decoding rule. So we make an error and decode $\boldsymbol{c}_o = c_N(a_o)$ as another letter $a \neq a_o$ only when either $d(\boldsymbol{c}_o, \boldsymbol{c}'_o) \geqslant r$ or $d(c_N(a), \boldsymbol{c}'_o) < r$ for some $a \neq a_o$. Hence the probability of an error is

$$\mathbb{P}(\text{ error }) \leqslant \frac{p(1-p)}{N(q-p)^2} \ + \ 2^{N(h(q)-h(p)-\varepsilon)}$$

because of (1) and (2). Choose q with

$$p < q < \tfrac{1}{2} \qquad \text{and} \qquad h(q) < h(p) + \varepsilon \ .$$

This is certainly possible for $0 \leqslant p < \frac{1}{2}$. Then

$$\frac{p(1-p)}{N(q-p)^2} \leqslant \tfrac{1}{2}\varepsilon \qquad \text{and} \qquad 2^{N(h(q)-h(p)-\varepsilon)} \leqslant \tfrac{1}{2}\varepsilon$$

for all sufficiently large N. For such N we see that the probability of error is strictly less than ε.

This is the error averaged over all of the random choices of code words. This means that there must be at least one choice of the code words that also has $\mathbb{P}(\text{ error }) < \varepsilon$. Take this as the code c_N.

We have almost proved the result we sought except that we have proved that the mean error $\mathbb{P}(\text{ error })$ rather than that the maximum error $\widehat{e}(c_N) = \max\{\mathbb{P}(\text{ error } \mid a \text{ sent }) : a \in \mathcal{A}\}$ is less than ε.

For our code c_N the average error probability satisfies

$$\frac{1}{|\mathcal{A}_N|} \sum_{a \in \mathcal{A}_N} \mathbb{P}(\text{ error } \mid a \text{ sent }) \ < \varepsilon$$

so, for at least half of the letters in $\mathcal{A}_N$ we must have

$$\mathbb{P}(\text{ error } \mid a \text{ sent }) < 2\varepsilon \ .$$

Choose a new alphabet $\mathcal{A}'_N$ that consists just of these $\frac{1}{2}K_N$ letters. Then we certainly have

$$\mathbb{P}(\text{ error } \mid a \text{ sent }) < 2\varepsilon \qquad \text{for } a \in \mathcal{A}'_N$$

so the maximum error is at most 2ε. Furthermore,

$$|\mathcal{A}'_N| = \tfrac{1}{2}|\mathcal{A}_N| = 2^{N(\mathrm{Cap}-\varepsilon)-1},$$

so the rate of transmission is

$$\frac{\log_2 |\mathcal{A}'_N|}{N} = \mathrm{Cap} - \varepsilon - \frac{1}{N} \ .$$

□

10.4 * Typical Sequences *

In this section we want to reinterpret the entropy $H(A)$ in terms of the probability of a "typical sequence" of values arising from a sequence of independent random variables (A_n) all with the same distribution as A.

As a motivating example, consider tossing an unfair coin with probability p of heads. Let A, A_n be the results of independent coin tosses. If we toss this coin a large number N times, we expect to get approximately pN heads and $(1-p)N$ tails. The probability of any particular sequence of pN heads and $(1-p)N$ tails is

$$p^{pN}(1-p)^{(1-p)N} = 2^{N(p\log_2 p+(1-p)\log_2(1-p))} = 2^{-NH(A)} .$$

Of course not every sequence of coin tosses will be like this. It is quite possible to get extraordinary sequences like all N heads. However, there is only a small probability of getting such an atypical sequence. With high probability we will get a typical sequence and its probability will be close to $2^{-NH(A)}$.

To make this precise we need to recall the weak law of large numbers from the probability course. This follows from Chebyshev's inequality.

We say that a sequence of random variables S_n *converge to a random variable L in probability* if

$$\mathbb{P}\left(|S_n - L| \geqslant \varepsilon\right) \to 0 \qquad \text{as} \qquad n \to \infty$$

for every $\varepsilon > 0$. This means that S_n and L can still take very different values for large n but only on a set with small probability.

Theorem 10.3 The weak law of large numbers
Let X be a discrete real-valued random variable. Let (X_n) be independent random variables each with the same distribution as X. The averages:

$$S_n = \frac{X_1 + X_2 + \ldots + X_n}{n}$$

then converge in probability to the constant $\mathbb{E}X$ as $n \to \infty$.

Proof:
We need to show that

$$\mathbb{P}\left(|S_n - \mathbb{E}X| \geqslant t\right) \to 0 \qquad \text{as} \qquad n \to \infty .$$

Note that

$$\mathbb{E}S_n = \frac{\mathbb{E}X_1 + \mathbb{E}X_2 + \ldots + \mathbb{E}X_n}{n} = \mathbb{E}X$$

and

$$\operatorname{var}(S_n) = \frac{\operatorname{var}(X_1) + \operatorname{var}(X_2) + \ldots + \operatorname{var}(X_n)}{n^2} = \frac{\operatorname{var}(X)}{n} .$$

Chebyshev's inequality gives

$$\mathbb{P}\left(|S_n - \mathbb{E}S_n| \geqslant t\right) \leqslant \frac{\operatorname{var}(S_n)}{t^2}$$

so we have

$$\mathbb{P}\left(|S_n - \mathbb{E}X| \geqslant t\right) \leqslant \frac{\operatorname{var} X}{nt^2}$$

which certainly tends to 0 as $n \to \infty$. □

Let A be a random variable that takes values in the finite set $\mathcal{A}$. Let (A_n) be a sequence of independent random variables each with the same distribution as A. Recall that the entropy of A is

$$H(A) = -\sum_{a\in\mathcal{A}} \mathbb{P}(A=a)\log_2 \mathbb{P}(A=a) \ .$$

This is the expectation of the real-valued random variable:

$$X(\omega) = -\log_2 \mathbb{P}(A = A(\omega)) \ .$$

(In other words, if A takes the values a_k with probabilities p_k, then X takes the values $-\log_2 p_k$ with the same probabilities.) In a similar way we define X_n from A_n, so the random variables X_n are independent and each has the same distribution as X.

The weak law of large numbers Theorem 10.3 shows that

$$\frac{X_1+X_2+\ldots X_n}{n} \to \mathbb{E}X \qquad \text{in probability as } n\to\infty \ .$$

Hence, for any $\varepsilon > 0$, there is an $N(\varepsilon)$ with

$$\mathbb{P}\left(\left|\frac{X_1+X_2+\ldots X_n}{n} - \mathbb{E}X\right| \geqslant \varepsilon\right) \leqslant \varepsilon$$

for $n \geqslant N(\varepsilon)$.

Consider a particular sequence of values $(a_j)_{j=1}^n$ from $\mathcal{A}$. The probability of obtaining this sequence as (A_j) is

$$\mathbb{P}\left(A_j = a_j \text{ for } j=1,2,\ldots,n\right) = \prod_{j=1}^n \mathbb{P}(A=a_j) \ .$$

Also, if we do have $A_j = a_j$, then

$$\frac{X_1+X_2+\ldots X_n}{n} = -\sum_{j=1}^n \log_2 \mathbb{P}(A=a_j) = -\log_2 \prod_{j=1}^n \mathbb{P}(A=a_j)$$

and therefore,

$$\left|\frac{X_1+X_2+\ldots X_n}{n} - \mathbb{E}X\right| < \varepsilon$$

is equivalent to

$$2^{-n(H(A)+\varepsilon)} < \mathbb{P}\left(A_j = a_j \text{ for } j=1,2,\ldots,n\right) < 2^{-n(H(A)-\varepsilon)} \ . \qquad *$$

Thus we see that the probability of obtaining a sequence (a_j) for which $(*)$ holds is at least $1-\varepsilon$ for $n \geqslant N(\varepsilon)$. These are the "typical sequences". Thus we have proved:

Theorem 10.4 Asymptotic equipartition property
Let A be a random variable taking values in the finite set $\mathcal{A}$ and with entropy $H(A)$. Let (A_n) be a sequence of independent random variables each with the same distribution as A. For each $\varepsilon > 0$, there is a natural number $N(\varepsilon)$ such that the following conditions hold for any $n \geqslant N(\varepsilon)$.

There is a set T of sequences in $\mathcal{A}^n$ with

$$\mathbb{P}((A_j)_{j=1}^n \in T) > 1-\varepsilon$$

and, for each sequence $(a_j) \in T$,

$$2^{-n(H(A)+\varepsilon)} < \mathbb{P}\left(A_j = a_j \text{ for } j=1,2,\ldots,n\right) < 2^{-n(H(A)-\varepsilon)} \ . \qquad *$$

We call the sequences in T the "typical ones" while those outside are "atypical". The sequences in T all have approximately the same probability $2^{-nH(A)}$ of arising, so we say that the sequence of random variable (A_n) has the *asymptotic equipartition property.*

11: LINEAR CODES

It is useful to use the algebra you have learnt in earlier courses to describe codes. In particular, if we are dealing with an alphabet of size $q = p^k$ for some prime number p, then there is a field $\mathbb{F}_q$ with exactly q elements and we can use this to represent the alphabet. The most important case is when $q = 2$, or a power of 2, and we have seen this already when we looked at Hamming's code.

11.1 Finite Fields

In this section we will recall results from the Numbers and Sets, Linear Algebra, and Groups, Rings and Modules courses. For this course, it will be sufficient to use only the integers modulo 2, $\mathbb{F}_2$ or occasionally the fields $\mathbb{F}_{2^K}$ for $K > 1$.

There are fields with a finite number q of elements if and only if q is a prime power $q = p^r$. These fields are unique, up to isomorphism, and will be denoted by $\mathbb{F}_q$.

Example:
For each prime number p, the integers modulo p form a field. So $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$.

The non-zero elements of a finite field $\mathbb{F}_q$ form a commutative group denoted by $\mathbb{F}_q^\times$. This is a cyclic group of order $q - 1$. Any generator of $\mathbb{F}_q^\times$ is called a *primitive element* for $\mathbb{F}_q$. Let α be a primitive element. Then the other elements of the group $\mathbb{F}_q^\times$ are the powers α^k for $k = 1, 2, 3, \ldots, q - 1$. These are all distinct. The order of the element α^k is $\dfrac{q-1}{(q-1,k)}$, so the number of primitive elements is given by Euler's totient function

$$\varphi(q-1) = |\{k : k = 1, 2, 3, \ldots, q-1 \text{ and } (q-1, k) = 1\}| \ .$$

Example:
For the integers modulo 7 we have 3 and 5 as the only primitive elements.

We will want to work with alphabets that are finite dimensional vector spaces over the finite fields $\mathbb{F}_q$. For example, $\mathbb{F}_q^N$ is an N dimensional vector space over $\mathbb{F}_q$. Any N-dimensional vector space has q^N vectors. The *scalar product* on $\mathbb{F}_q^N$ is given by

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{n=1}^{N} x_n y_n \ .$$

(Every linear map $A : \mathbb{F}_q^N \to \mathbb{F}_q$ is of the form $\boldsymbol{x} \mapsto \boldsymbol{x} \cdot \boldsymbol{y}$ for some $\boldsymbol{y} \in \mathbb{F}_q^N$. So the scalar product gives us a way to identify the dual space $(\mathbb{F}_q^N)^*$ with $\mathbb{F}_q^N$.) For any finite set S, the set of all functions $V = \mathbb{F}_q^S = \{f : S \to \mathbb{F}_q\}$ is a vector space over $\mathbb{F}_q$ with dimension $|S|$.

A *polynomial* over $\mathbb{F}_q$ is a formal expression:

$$A(X) = a_0 + a_1 X + a_2 X^2 + \ldots + a_D X^D$$

with the coefficients $a_n \in \mathbb{F}_q$. In this expression, X is an indeterminate or formal parameter rather than an element of the field. The *degree* $\deg(A)$ of the polynomial A is the largest n with $a_n \neq 0$. The degree of the 0 polynomial is defined to be $-\infty$. Define addition and multiplication of polynomials as usual. This makes the set $\mathbb{F}_q[X]$ of all polynomials over $\mathbb{F}_q$ a ring. The degree is an Euclidean function for this ring and makes it an Euclidean domain: For polynomials $A, D \in \mathbb{F}_q[X]$ with $D \neq 0$, there are uniquely determined polynomials Q and R with

$$A = Q.D + R \qquad \text{and} \qquad \deg(R) < \deg(D) \ .$$

This has many useful consequences. We have the division criterion:

$$(X-\alpha)|A \qquad \Leftrightarrow \qquad A(\alpha)=0$$

for an element $\alpha \in \mathbb{F}_q$. Furthermore, the polynomial ring is a principal ideal domain. For, if I is an ideal in $\mathbb{F}_q[X]$ (written $I \lhd \mathbb{F}_q[X]$), then either $I=\{0\}$ or there is a non-zero polynomial $D \in I$ of smallest degree. For any other polynomial $A \in I$ we can write

$$A = Q.D + R \qquad \text{with} \qquad \deg(R) < \deg(D)\ .$$

However, $R \in I$, so R must be the 0 polynomial. Therefore $A = Q.D$. Hence every ideal is of the form $D\mathbb{F}_q[X]$ for some polynomial D. We will denote this ideal by (D).

For any ideal $I = D\mathbb{F}_q[X]$, the quotient $\mathbb{F}_q[X]/I$ is a ring and there is a quotient ring homomorphism $\mathbb{F}_q[X] \to \mathbb{F}_q[X]/I$. The quotient is a vector space over $\mathbb{F}_q$ with dimension $\deg(D)$.

A very important example of this is when D is the polynomial X^n-1 and we consider the quotient $\mathbb{F}_q[X]/(X^n-1)$. If we divide any polynomial $A \in \mathbb{F}_q[X]$ by X^n-1 we obtain a remainder of degree strictly less than n. So each coset $A + (X^n-1)\mathbb{F}_q[X]$ contains an unique polynomial R of degree at most $n-1$. Hence we can represent the quotient $\mathbb{F}_q[X]/(X^n-1)$ by

$$\{a_0 + a_1X + \ldots + a_{n-1}X^n : a_0, a_1, \ldots, a_{n-1} \in \mathbb{F}_q\}\ .$$

Multiplication in the quotient $\mathbb{F}_q[X]/(X^n-1)$ then corresponds to multiplying the polynomials and reducing the result modulo X^n-1, that is replacing any power X^k for $k \geqslant n$ by X^m where $k \equiv m \pmod{n}$. Note that, in this case, the quotient $\mathbb{F}_q[X]/(X^n-1)$ is a vector space of degree n.

11.2 Linear Codes

Suppose that we are looking at codes $c : \mathcal{A} \to \mathcal{B}^M$. If both of the alphabets $\mathcal{A}$ and $\mathcal{B}$ have orders that are powers of the same prime power q, then we can represent both as vector spaces over $\mathbb{F}_q$. So c is a map $\mathbb{F}_q^K \to \mathbb{F}_q^N$ for suitable dimensions K, N. Such a map is much easier to specify if it is linear, for then we need only give the images of a basis for $\mathbb{F}_q^K$. This is in itself a considerable advantage and makes it much simpler to implement the encoding and decoding of messages.

We will write $\mathbb{F}$ for any finite field, so $\mathbb{F} = \mathbb{F}_q$ for some prime power q. A code is *linear* if it is an injective linear map $c : \mathbb{F}^K \to \mathbb{F}^N$. The image of c is then a vector subspace of $\mathbb{F}^N$, called the *code book* of c. Its dimension is the rank of c.

Example:
Hamming's code was a linear map $\mathbb{F}_2^4 \to \mathbb{F}_2^7$.

A linear code $c : \mathbb{F}^K \to \mathbb{F}^N$ is given by an $N \times K$ matrix C with entries in $\mathbb{F}$. So $c : \boldsymbol{x} \mapsto C\boldsymbol{x}$. Since c is injective, the matrix C has nullity 0 and rank K. As usual, we are really only interested in the code book. This can be specified by giving any basis for the image.

The columns of the matrix C are one basis of code words for the code book. We can apply column operations to these to reduce the matrix to echelon form. For example

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

where $*$ denotes an arbitrary but unimportant value. Applying column operations to this we can obtain a matrix:

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

The columns of this matrix are still a basis for the code book. If we re-order the rows of this matrix, which corresponds to re-ordering the components of $\mathbb{F}^N$, then we obtain a matrix of the form

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

In general, when we apply these operations to our code matrix C we change it to the form

$$C = \begin{pmatrix} I \\ A \end{pmatrix}$$

where I is the $K \times K$ identity matrix and A is an arbitrary $(N-K) \times K$ matrix. Whenever we need to we can reduce our linear code to this standard form. When we have a matrix of this form, the code is

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \\ \sum a_{1j} x_j \\ \vdots \\ \sum a_{(N-K)j} x_j \end{pmatrix} \qquad \text{or, more concisely,} \qquad \boldsymbol{x} \mapsto \begin{pmatrix} \boldsymbol{x} \\ A\boldsymbol{x} \end{pmatrix} .$$

So the first K terms carry the information about the vector $\boldsymbol{x}$ while the remainder are check digits.

11.3 The Syndrome

Let $c : \mathbb{F}^K \to \mathbb{F}^N$ be a linear code. Its image is a vector subspace of $\mathbb{F}^N$ with dimension K. So we can construct a linear map

$$s : \mathbb{F}^N \to \mathbb{F}^{N-K} \qquad \text{with } \ker(s) = \mathrm{Im}(c) .$$

Such a linear map is called a *syndrome of c*.

For example, if we write the matrix C for c in the standard form $C = \begin{pmatrix} I \\ A \end{pmatrix}$, then the matrix $S = (\,-A \quad I\,)$ satisfies

$$\ker(S) = \left\{ \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} : -A\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{0} \right\} = \left\{ \begin{pmatrix} \boldsymbol{u} \\ A\boldsymbol{u} \end{pmatrix} : \boldsymbol{u} \in \mathbb{F}^K \right\} = \mathrm{Im}(C) .$$

So S is a syndrome for the code.

The syndrome gives an easy way to check whether a vector $\boldsymbol{x} \in \mathbb{F}^N$ is a code word, for we simply compute $S\boldsymbol{x}$ and see whether it is $\mathbf{0}$. We can also define a code by giving a syndrome, for the code book is then $\ker(s)$.

Exercise:
For a natural number d there are $N = 2^d - 1$ non-zero vectors in $\mathbb{F}_2^d$. Take these as the columns of an $d \times N$ matrix S. The kernel of S is then the code book for a Hamming code

$$c : \mathbb{F}_2^{N-d} \to \mathbb{F}_2^N \ .$$

Check that the case $d = 3$ gives the Hamming code we constructed earlier.

Show that each of these Hamming codes is a perfect 1-error correcting code.

(The code $c : \mathbb{F}^K \to \mathbb{F}^N$ and the syndrome $s : \mathbb{F}^N \to \mathbb{F}^{N-K}$ are really dual to one another. For the dual maps are

$$s^* : \mathbb{F}^{N-K} \to \mathbb{F}^N \qquad \text{and} \qquad c^* : \mathbb{F}^N \to \mathbb{F}^K$$

with $\ker(c^*) = \operatorname{Im} c^\perp = \ker(s)^\perp = \operatorname{Im}(s^*)$. So s^* is a linear code with c^* as its syndrome. These correspond to transposing the matrices. So $S^\top$ is the matrix for a linear code with $C^\top$ its syndrome.)

The minimum distance for a code is the minimum Hamming distance between two different code words. When the code is linear we have $d(\boldsymbol{c}', \boldsymbol{c}) = d(\mathbf{0}, \boldsymbol{c} - \boldsymbol{c}')$. So the minimum distance is

$$\min \{ d(\mathbf{0}, \boldsymbol{c}) : \boldsymbol{c} \text{ is a non-zero code word } \} \ .$$

The *weight of a code word* $\boldsymbol{x}$ is the Hamming distance of $\boldsymbol{x}$ from $\mathbf{0}$. So the minimum distance for a linear code is the minimum weight for non-zero code words.

We will use minimum distance decoding. Suppose that we receive a word $\boldsymbol{x} \in \mathbb{F}^N$. We compute its syndrome $s(\boldsymbol{x})$. If this is $\mathbf{0}$ then we know that $\boldsymbol{x}$ is a code word. Otherwise there have been errors in transmission and we need to find the code word closest to $\boldsymbol{x}$.

We proceed as follows. For each vector $\boldsymbol{y} \in \mathbb{F}^{N-K}$, find a vector $\boldsymbol{z} \in s^{-1}(\boldsymbol{y})$ with minimum weight and call this $u(\boldsymbol{y})$. Note that we can do this for every vector $\boldsymbol{y} \in \mathbb{F}^{N-K}$ in advance of receiving any message, at least when $\mathbb{F}^{N-K}$ is not too large. It is clear that $s(u(\boldsymbol{y})) = \boldsymbol{y}$.

Now we decode the received vector $\boldsymbol{x}$ as $\boldsymbol{c}_o = \boldsymbol{x} - u(s(\boldsymbol{x}))$. Since

$$s(\boldsymbol{x} - u(s(\boldsymbol{x}))) = s(\boldsymbol{x}) - s(u(s(\boldsymbol{x}))) = \mathbf{0}$$

we see that $\boldsymbol{c}_o$ is indeed a code word. For any other code word $\boldsymbol{c}$, we have $d(\boldsymbol{c}, \boldsymbol{x}) = d(\mathbf{0}, \boldsymbol{x} - \boldsymbol{c})$. However, $s(\boldsymbol{x} - \boldsymbol{c}) = s(\boldsymbol{x})$, so the definition of u ensures that

$$d(\mathbf{0}, \boldsymbol{x} - \boldsymbol{c}) \geqslant d(\mathbf{0}, u(s(\boldsymbol{x}))) = d(\mathbf{0}, \boldsymbol{x} - \boldsymbol{c}_o)$$

and hence that

$$d(\boldsymbol{c}, \boldsymbol{x}) \geqslant d(\boldsymbol{c}_o, \boldsymbol{x}) \ .$$

So $\boldsymbol{c}_o$ is the code word closest to $\boldsymbol{x}$.

11.4 Reed – Muller Codes

In this section we will construct linear codes that have proved useful in dealing with very noisy channels.

Let S be the finite dimensional vector space $\mathbb{F}_2^M$ over the field $\mathbb{F}_2$ of integers modulo 2. This is itself a finite set with 2^M elements. Let V be the set of all functions from S to $\mathbb{F}_2$:

$$V = \mathbb{F}_2^S = \{f : S \to \mathbb{F}_2\} \ .$$

The functions

$$1_{\boldsymbol{y}}(\boldsymbol{x}) = \begin{cases} 1 & \text{when } \boldsymbol{x} = \boldsymbol{y}; \\ 0 & \text{otherwise} \end{cases}$$

are the standard basis for V, so V has dimension 2^M. We will use this basis to determine the Hamming distance in V, so it is

$$d(f,g) = |\{\boldsymbol{x} \in S : f(\boldsymbol{x}) \neq g(\boldsymbol{x})\}| \ .$$

Let π_i be the function $\boldsymbol{x} \mapsto x_i$ that maps each vector in S to its ith component. Let I be a subset $\{i(1), i(2), \dots, i(r)\}$ of $\{1,2,3,\dots,M\}$ and write π_I as an abbreviation for the function

$$\boldsymbol{x} \mapsto \pi_{i(1)}(\boldsymbol{x})\pi_{i(2)}(\boldsymbol{x}) \dots \pi_{i(r)}(\boldsymbol{x}) = \prod_{i \in I} x_i \ .$$

This is a function in V. In particular the empty sequence $\emptyset$ has $\pi_\emptyset$ equal to the constant function 1.

Lemma 11.1
The functions π_I for $I \subset \{1,2,3,\dots,M\}$ are a basis for V.

Proof:
We can write the value of $1_{\boldsymbol{y}}(\boldsymbol{x})$ as the product

$$1_{\boldsymbol{y}}(\boldsymbol{x}) = \prod_{i=1}^{M} (x_i - y_i + 1) \ .$$

Now expand this product to get

$$1_{\boldsymbol{y}}(\boldsymbol{x}) = \sum_I \alpha_I \left(\prod_{i \in I} x_i \right)$$

for some scalar coefficients $\alpha_I \in \mathbb{F}_2$ that depend on $\boldsymbol{y}$. This shows that $1_{\boldsymbol{y}}$ is the linear combination $1_{\boldsymbol{y}} = \sum_I \alpha_I \pi_I$ of the functions π_I.

Since $(1_{\boldsymbol{y}})$ is a basis for V, we see that the functions π_I span V. There are $2^M = \dim V$ of the π_I, so they must form a basis for V. □

The lemma shows that we can write any function $f \in V$ as a linear combination

$$f = \sum_I a_I \pi_I \ .$$

The *Reed – Muller code $RM(M,r)$* has a code book equal to

$$\left\{ \sum (\alpha_I \pi_I : |I| \leqslant r) : \alpha_I \in \mathbb{F}_2 \right\} \ .$$

This code book is a vector subspace with dimension

$$\binom{M}{0} + \binom{M}{1} + \dots \binom{M}{r}$$

so it gives a linear code of this rank and length $\dim V = 2^M$.

Example:
The $RM(M, 0)$ code has only two code words 0 and 1 so it is the repetition code that repeats each bit 2^M times. The Hamming distance $d(0, 1)$ is 2^M, so this is also the minimum distance for $RM(M, 0)$.

$RM(M, 1)$ has dimension $1 + M$ and has the functions π_j for $j = 1, 2, 3, \dots, M$ as a basis. The Hamming distance satisfies

$$d(0, \pi_j) = 2^{M-1}$$

and it is easy to see that this is the minimum distance for $RM(M, 1)$.

Proposition 11.2 Reed – Muller codes
The Reed – Muller code $RM(M, r)$ has minimum distance 2^{M-r} for $0 \leqslant r \leqslant M$.

Proof:
We need to show that the minimum weight of a non-zero code word in $RM(M, r)$ is 2^{M-r}. We will do this by induction on M. We already know the result for $M = 1$.

Suppose that we have already established the result for $M - 1$ and all r.

Note first that, for the set $J = \{1, 2, \dots, r\}$ we have $d(0, \pi_J) = 2^{M-r}$ so the minimum distance is at most this large. Let $f = \sum_I \alpha_I \pi_I$ be a non-zero function in $RM(M, r)$. We split this into two parts depending on whether I contains M or not. So

$$f = \left(\sum_{M \notin I} \alpha_I \pi_I \right) + \left(\sum_{M \in I} \alpha_I \pi_{I \setminus \{M\}} \right) \pi_M = f_0 + f_1.\pi_M \ .$$

The functions f_0 and f_1 do not depend on x_M so we think of them as functions on $\mathbb{F}_2^{M-1}$. The sum f_0 is in $RM(M - 1, r)$ while $f_1 \in RM(M - 1, r - 1)$.

Denote by d_M the Hamming distance in $V = \mathbb{F}_2^M$ and d_{M-1} the Hamming distance in $\mathbb{F}_2^{M-1}$. Observe that

$$\begin{array}{ll} f(x_1, x_2, \dots, x_{M-1}, 0) = f_0(x_1, x_2, \dots, x_{M-1}) & \text{on the set where } x_M = 0; \\ f(x_1, x_2, \dots, x_{M-1}, 1) = (f_0 + f_1)(x_1, x_2, \dots, x_{M-1}) & \text{on the set where } x_M = 1. \end{array}$$

Since f is non-zero, either f_0 or $f_0 + f_1$ must be non-zero. If $f_0 = 0$, then f_1 is non-zero and

$$d_M(0, f) = d_{M-1}(0, f_1) \geqslant 2^{(M-1)-(r-1)} \ .$$

If $f_0 \neq 0$ and $f_1 = -f_0$, then $f_0 = -f_1 \in RM(M - 1, r - 1)$ so we have

$$d_M(0, f) = d_{M-1}(0, f_0) \geqslant 2^{(M-1)-(r-1)} \ .$$

Finally, if $f_0 \neq 0$ and $f_1 \neq -f_0$, then

$$d_M(0, f) = d_{M-1}(0, f_0) + d_{M-1}(0, f_0 + f_1) \geqslant 2 \times 2^{M-1-r}$$

since both f_0 and $f_0 + f_1$ are non-zero elements of $RM(M - 1, r)$. In all cases we have

$$d_M(0, f) \geqslant 2^{M-r}$$

which completes the inductive step. □

The Reed – Muller $RM(5, 1)$ code was used by NASA in the Mariner mission to Mars. This code has rank $1 + 5 = 6$ and length $2^5 = 32$. So its rate of tranmission is $\frac{6}{32} = \frac{3}{16}$. The last proposition show that it has minimum distance $2^4 = 16$. So it is 15-error detecting and 7-error correcting.

12: CYCLIC CODES

A linear code is *cyclic* if $c_N c_1 c_2 \dots c_{N-2} c_{N-1}$ is a code word whenever $c_1 c_2 c_3 \dots c_{N-1} c_N$ is a code word. This means that the *shift map*:

$$T : \mathbb{F}^N \to \mathbb{F}^N \ ; \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} \mapsto \begin{pmatrix} x_N \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

maps the code book into itself.

Example:
The matrix

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

defines a cyclic code over $\mathbb{F}_2$ since, if we label the columns of C as $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$ and $\boldsymbol{c}_4$, then

$$T(\boldsymbol{c}_4) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \boldsymbol{c}_1 + \boldsymbol{c}_2 + \boldsymbol{c}_3 \ .$$

It is easier to describe cyclic codes if we identify $\mathbb{F}^N$ with the quotient $\mathbb{F}[X]/(X^N - 1)$. Recall that $\mathbb{F}[X]/(X^N - 1)$ is the set of equivalence classes of polynomials modulo $X^N - 1$. Given a polynomial $P(X)$, divide it by $X^N - 1$ to get $P(X) = Q(X)(X^N - 1) + R(X)$ with $\deg R(X) < N$. Then $P(X)$ is equivalent modulo $X^N - 1$ to

$$R(X) = r_0 + r_1 X + r_2 X^2 + \ldots + r_{N-1} X^{N-1} \in \{A(X) \in \mathbb{F}[x] : \deg A(X) < N\} \ .$$

We will write $[P(X)]$ for the equivalence class of $P(X)$ modulo $X^N - 1$. So we have seen how to identify $[P(X)]$ with $[R(X)]$ and hence with the vector

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{N-2} \\ r_{N-1} \end{pmatrix} \in \mathbb{F}^N \ .$$

The shift map corresponds to

$$T : \mathbb{F}[X]/(X^N - 1) \to \mathbb{F}[X]/(X^N - 1) \ ; \quad [P(X)] \mapsto [X.P(X)]$$

because $X.X^{N-1} = X^N \equiv 1 \pmod{X^N - 1}$. The quotient homomorphism will be denoted by

$$q : \mathbb{F}[X] \to \mathbb{F}[X]/(X^N - 1) \ ; \quad P(X) \mapsto [P(X)] \ .$$

Proposition 12.1 Cyclic codes
Let W be a vector subspace of $\mathbb{F}[X]/(X^N - 1)$ and let $J = q^{-1}(W)$. Then W is the code book for a cyclic code if and only if J is an ideal of $\mathbb{F}[X]$.

Proof:

The quotient map q is linear, so J is certainly a vector subspace of $\mathbb{F}[X]$. Suppose that W is cyclic. Then $[P(X)] \in W \Rightarrow [X.P(X)] \in W$ for any polynomial $P(X)$. This means that

$$P(X) \in J \Rightarrow X.P(X) \in J \ .$$

Therefore, if $P(X) \in J$, then

$$A(X)P(X) = \left(\sum a_k X^k\right) P(X) = \sum a_k X^K.P(X) \in J$$

for any polynomial $A(X)$. Hence J is an ideal of $\mathbb{F}[X]$: $J \lhd \mathbb{F}[X]$.

Conversely, suppose that $J \lhd \mathbb{F}[X]$ and that $[P(X)] \in W$. Then $P(X) \in J$ and so $X.P(X) \in J$. This means that $[X.P(X)] \in W$. So W is cyclic. $\square$

Corollary 12.2 Generators of cyclic codes
Let W be the code book for a cyclic code in $\mathbb{F}[X]/(X^N - 1)$. Then

$$W = \{[A(X)G(X)] : A(X) \in \mathbb{F}[X]\}$$

for a generator polynomial $G(X) \in \mathbb{F}[X]$. Moreover, $G(X)$ is a divisor of $X^N - 1$.

Proof:

The proposition shows that $J = q^{-1}(W)$ is an ideal of $\mathbb{F}[X]$. Since $\mathbb{F}[X]$ is a principal ideal domain, this ideal must be of the form $G(X)\mathbb{F}[X]$ for some polynomial $G(X)$. Therefore, $W = \{[A(X)G(X)] : A(X) \in \mathbb{F}[X]\}$.

Now $0 = q(X^N - 1)$ is in W, so $X^N - 1 \in J$. Hence $G(X)$ must divide $X^N - 1$. $\square$

We may always multiply the generator polynomial $G(X)$ by a scalar to ensure that it has leading coefficient 1. Then the generator polynomial is uniquely determined by the cyclic code book W. Since $G(X)$ divides $X^N - 1$, we may write $X^N - 1 = G(X)H(X)$. We call $H(X)$ the *check polynomial* for the code. Once again, it is unique up to a scalar multiple.

Proposition 12.3
Let $G(X)$ be the generator of a cyclic code of length N over $\mathbb{F}$. Then the vectors

$$[G(X)],\ [X.G(X)],\ [X^2.G(X)],\ \ldots,\ [X^{N-D-2}.G(X)],\ [X^{N-D-1}.G(X)]$$

are a basis for the code book in $\mathbb{F}[X]/(X^N - 1)$. Hence the code book has rank $N - D$ where $D = \deg G(X)$.

Proof:

We know that each of the products $X^k.G(X)$ is in the ideal J, so the vectors $[X^k.G(X)]$ must lie in the code book W for the cyclic code.

Every vector in W is of the form $[G(X)P(X)]$ for some polynomial $P(X) \in \mathbb{F}[X]$. By reducing $G(X).P(X)$ modulo $X^N - 1$ we may ensure that $\deg P(X) < N - D$. So any vector in W is of the form

$$p_0[G(X)] + p_1[X.G(X)] + p_2[X^2.G(X)] + \ldots + p_{N-D-1}[X^{N-D-1}.G(X)] \ .$$

Thus the vectors $[G(X)],\ [X.G(X)],\ [X^2.G(X)],\ \ldots,\ [X^{N-D-2}.G(X)],\ [X^{N-D-1}.G(X)]$ span W.

Suppose that these vectors were linearly dependent, say

$$p_0[G(X)] + p_1[X.G(X)] + p_2[X^2.G(X)] + \ldots + p_{N-D-1}[X^{N-D-1}.G(X)] = 0 \ .$$

Then

$$[(p_0 + p_1X + p_2X^2 + \ldots + p_{N-D-1}X^{N-D-1})G(X)] = [P(X)G(X)] = 0$$

so $(X^N - 1)|P(X)G(X)$. This means that $H(X)|P(X)$. Since $\deg P(X) < N - D = \deg H(X)$, this implies that $P(X) = 0$. Hence the vectors $[G(X)],\ [X.G(X)],\ [X^2.G(X)],\ \ldots,\ [X^{N-D-2}.G(X)],\ [X^{N-D-1}.G(X)]$ are linearly independent. $\square$

Using the basis for the code book in this proposition, we see that a matrix for the code is the $N \times (N-D)$ matrix:

$$C = \begin{pmatrix} g_0 & 0 & 0 & \dots & 0 \\ g_1 & g_0 & 0 & \dots & 0 \\ g_2 & g_1 & g_0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ g_D & g_{D-1} & g_{D-2} & \dots & \\ 0 & g_D & g_{D-1} & \dots & \\ 0 & 0 & g_D & \dots & \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & g_D \end{pmatrix}$$

where $G(X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_D X^D$.

Let S be the $D \times N$ matrix:

$$S = \begin{pmatrix} 0 & 0 & 0 & \dots & h_2 & h_1 & h_0 \\ 0 & 0 & 0 & \dots & h_1 & h_0 & 0 \\ 0 & 0 & 0 & \dots & h_0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \\ 0 & h_{N-D} & h_{N-D-1} & \dots & 0 & 0 & \\ h_{N-D} & h_{N-D-1} & h_{N-D-2} & \dots & 0 & 0 & \end{pmatrix}$$

where $H(X) = h_0 + h_1 X + h_2 X^2 + \ldots + h_{N-D} X^{N-D}$. Then $h_{N-D} \neq 0$, so the rows of S are linearly independent. So its rank is D and its nullity must be $N-D$. However, it is easy to check that each column of the matrix C lies in the kernel of S, because $G(X)H(X) = X^N - 1$ so

$$\sum_j h_{m-j} g_j = 0 \qquad \text{for } \ 0 < m < N \ .$$

This means that the kernel of S is spanned by the columns of C. So $\ker S = W = \mathrm{Im}(C)$ and S is a syndrome matrix for the cyclic code.

Example:
Over the field $\mathbb{F}_2$ of integers modulo 2 we have

$$X^7 - 1 = ((1 + X + X^3)(1 + X + X^2 + X^4) \ .$$

Let $G(X) = 1 + X + X^3$. This generates a cyclic binary code of length 7. A syndrome matrix is

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

All of the non-zero vectors in $\mathbb{F}_2^3$ occur as columns in this matrix, so the code is Hamming's code with the terms of the code re-ordered.

1

13: BCH CODES

13.1 The Characteristic of a Field

Let K be a finite field and consider the sequence

$$0_K,\ 1_K,\ 1_K+1_K,\ 1_K+1_K+1_K,\ \ 1_K+1_K+1_K+1_K,\ \dots\ .$$

Eventually this sequence must repeat, so we obtain a first natural number p with the sum of p terms $1_K + 1_K + \ldots + 1_K$ equal to 0. If p factorised, then one of the factors would also have this property, so p must be a prime number. It is the *characteristic of* K. The set of p elements $0_K, 1_K, 1_K+1_K$, $1_K+1_K+1_K$, ... is then a subfield of K which is isomorphic to the integers modulo p. Hence K contains $\mathbb{F}_p$. (K is then a vector space over $\mathbb{F}_p$ of some dimension r, so it has p^r elements. Thus K has characteristic p when it is one of the fields $\mathbb{F}_{p^r}$.)

We will only consider fields with characteristic 2 in this section, so they are all fields of order 2^r and will give us binary codes. The arguments readily extend to other characteristics.

We know that any cyclic code has a generator polynomial $G(X)$ with $G(X)|(X^N-1)$ where N is the length of the code. We will assume that N is odd. It can be shown (in the Galois Theory course) that there is some finite field K containing $\mathbb{F}$ in which X^N-1 factorises completely into linear factors (or "splits"). The next lemma shows when these factors are all distinct.

Lemma 13.1 Separable fields
Let N be an odd natural number and K a field with characteristic 2 in which X^N-1 factorises completely into N linear factors. Then X^N-1 has N distinct roots in K. These form a cyclic group.

Proof:
Suppose that X^N-1 had a repeated root α in K. Then

$$X^N-1=(X-\alpha)^2P(X)$$

for some polynomial $P(X)\in K[X]$. Now if we differentiate this formally we obtain

$$NX^{N-1}=2(X-\alpha)P(X)+(X-\alpha)^2P'(X)=(X-\alpha)\left(2P(X)+(X-\alpha)P'(X)\right)\ .$$

So $X-\alpha$ divides both X^N-1 and NX^{N-1}. Since N is odd, $N1_K\neq 0_K$. So $X-\alpha$ must divide the highest common factor of X^N-1 and X^{N-1}. This highest common factor is 1, so we get a contradiction.

We have now shown that the set S of roots of X^N-1 in K contains N distinct elements. It is clearly a subgroup of the group $K^\times$. We know that $K^\times$ is cyclic, so the subgroup S must be also. □

The lemma shows that we can find a *primitive root* α *of* X^N-1 in the field K with the other roots being

$$\alpha,\alpha^2,\alpha^3,\ldots,\alpha^{N-1}\ .$$

The lemma is not true if N is even. For example, $X^2-1=(X-1)^2$.

13.2 BCH codes

Let $\mathbb{F}$ be a finite field with characteristic 2. Choose an odd integer N and let K be a field containing $\mathbb{F}$ in which $X^N - 1$ factorises completely into linear factors. Take α as a primitive root of $X^N - 1$ in the field K. The Bose - Ray Chaudhuri – Hocquenghem (BCH) code with design distance δ is the cyclic code of length N over $\mathbb{F}$ defined by the ideal

$$J = \{P(X) \in \mathbb{F}[X] : P(\alpha) = P(\alpha^2) = P(\alpha^3) = \ldots = P(\alpha^{\delta-1}) = 0\} .$$

Here δ is a natural number with $1 \leqslant \delta \leqslant N$.

The generating polynomial $G(X)$ for this code is the minimal polynomial in $\mathbb{F}[X]$ that is 0 at each of the points $\alpha, \alpha^2, \ldots, \alpha^{\delta-1}$ in K. It is thus the least common multiple of the minimal polynomials for these points. It is clearly a factor of $X^N - 1$. So the BCH code is a cyclic linear code of length N.

Example:
Consider the polynomial $X^7 - 1$ over $\mathbb{F}_2$. The roots of this form a cyclic group of order 7, so every root is a primitive root. We can factorise $X^7 - 1$ over $\mathbb{F}_2$ as

$$X^7 - 1 = (X-1)(X^3 + X^2 + 1)(X^3 + X + 1) .$$

The cubic factors are irreducible over $\mathbb{F}_2$ since, if they factorized further, then one of the factors would be linear and so give a root in $\mathbb{F}_2$.

Let α be one of the roots of $X^3 + X + 1$ in K. Then $\alpha^7 = 1$ and so

$$(\alpha^2)^3 + (\alpha^2) + 1 = \alpha^6 + \alpha^2 + 1 = \alpha^6(1 + \alpha^3 + \alpha) = 0 .$$

This shows that α^2 is also a root of $X^3 + X + 1$. Repeating the argument shows that $\alpha^4 = (\alpha^2)^2$ is a root. These roots are distinct, so

$$X^3 + X + 1 = (X - \alpha)(X - \alpha^2)(X - \alpha^4) .$$

Similarly we see that

$$X^3 + X^2 + 1 = (X - \alpha^3)(X - \alpha^5)(X - \alpha^6) .$$

The BCH code with design distance 3 is given by the ideal

$$J = \{P(X) \in \mathbb{F}_2[X] : P(\alpha) = P(\alpha^2) = 0\} .$$

The generating polynomial for this ideal is $X^3 + X + 1$. We saw at the end of lecture 12 that this gave a rearrangement of Hamming's code.

We want to prove that the minimum distance for a BCH code is at least as big as the design distance δ. To do this we need to recall a result about determinants.

Lemma 13.2 van der Monde determinants
The determinant

$$\Delta(X_1, X_2, \ldots, X_K) = \begin{vmatrix} X_1 & X_2 & X_3 & \ldots & X_K \\ X_1^2 & X_2^2 & X_3^2 & \ldots & X_K^2 \\ X_1^3 & X_2^3 & X_3^3 & \ldots & X_K^3 \\ \vdots & \vdots & \vdots & & \vdots \\ X_1^K & X_2^K & X_3^K & \ldots & X_K^K \end{vmatrix}$$

satisfies

$$\Delta(X_1, X_2, \ldots, X_K) = \left(\prod_{k=0}^{K} X_k \right) \left(\prod_{i<j} (X_j - X_i) \right) .$$

Proof:

We will treat the (X_k) as independent indeterminants and prove the lemma by induction on K. The result is clearly true for $K = 1$.

Consider $\Delta(X_1, X_2, \dots, X_K)$ as a polynomial of degree K in X_K with coefficients that are polynomials in $X_1, X_2, \dots, X_{K-1}$. When $X_K = 0$ or X_1 or X_2 or $\dots$ or X_{K-1}, then the determinant is obviously 0, so

$$X_K\left(\prod_{i<K}(X_K - X_i)\right)$$

must divide the determinant. Hence

$$\Delta(X_1, X_2, \dots, X_K) = c(X_1, X_2, \dots, X_{K-1})X_K\left(\prod_{i<K}(X_K - X_i)\right)$$

for some polynomial $c(X_1, X_2, \dots, X_{K-1})$. The leading coefficients of both sides of this equation are

$$\Delta(X_1, X_2, \dots, X_{K-1}) \qquad \text{and} \qquad c(X_1, X_2, \dots, X_{K-1}) \ .$$

So we see that

$$\Delta(X_1, X_2, \dots, X_K) = \Delta(X_1, X_2, \dots, X_{K-1})X_K\left(\prod_{i<K}(X_K - X_i)\right)$$

which completes the induction. □

Theorem 13.3 Minimum distance for BCH codes
The minimum distance for a BCH code with design distance δ is at least δ.

Proof:

Let $P(X) = p_0 + p_1X + \dots + p_{N-1}X^{N-1}$ be a non-zero polynomial in the code book. Then we have

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \alpha^{N-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \dots & \alpha^{2(N-1)} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \dots & \alpha^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{\delta-1} & \alpha^{2(\delta-1)} & \alpha^{3(\delta-1)} & \dots & \alpha^{(N-1)(\delta-1)} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{N-2} \\ p_{N-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} .$$

Call the matrix above A. Lemma 13.1 shows that any $\delta - 1$ columns of A are linearly independent because no two of the N powers of α are equal or 0. This means that there must be at least δ non-zero coefficients (p_k) in order for $A\boldsymbol{p}$ to be $\mathbf{0}$. Hence the minimum distance is at least δ. □

It remains for us to consider how to decode a message. Since our BCH code is a linear code, we can use the general methods for finding the closest code word to any message we receive. However, we can also exploit the algebraic definition of BCH codes to find more efficient ways to do this.

We know from Proposition 7.3 that our BCH code is t-error correcting, where $t = \lfloor \frac{1}{2}(\delta - 1) \rfloor$. So suppose that a code word $\boldsymbol{c}$ is sent and we receive $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ where the error vector $\boldsymbol{e}$ has at most t non-zero entries. How do we find $\boldsymbol{e}$ and thence recover the sent code word $\boldsymbol{c}$?

Let $C(X), R(X)$ and $E(X)$ be the polynomials with degree less than N that correspond to the vectors $\boldsymbol{c}, \boldsymbol{r}$ and $\boldsymbol{e}$. We know that the code word $C(X)$ satisfies

$$C(\alpha) = C(\alpha^2) = \dots = C(\alpha^{\delta-1}) = 0 \ .$$

So
$$R(\alpha) = E(\alpha)\ ,\quad R(\alpha^2) = E(\alpha^2)\ ,\quad \ldots, R(\alpha^{\delta-1}) = E(\alpha^{\delta-1})\ .$$

First calculate $R(\alpha^j)$ for $j = 1, 2, \ldots, \delta - 1$. If these are all 0 then $R(X)$ is a code word and there have been no errors (or at least δ of them).

Otherwise let $\mathcal{E} = \{i : e_i \neq 0\}$ be the set of indices at which errors occur and assume that $0 < |\mathcal{E}| \leqslant t$. The *error locator polynomial* is
$$\sigma(X) = \prod_{i\in\mathcal{E}}(1 - \alpha^i X)\ .$$
This is a polynomial (over K) of degree $|\mathcal{E}|$ with constant term 1. If we know $\sigma(X)$ then we can easily find which powers α^{-i} are roots of $\sigma(X)$ and hence find the indices where errors have occurred. We could then correct the errors by changing those entries. So we need to calculate $\sigma(X)$.

Consider the formal power series
$$\eta(X) = \sum_{j=1}^{\infty} E(\alpha^j)X^j\ .$$
(Since $\alpha^N = 1$, the coefficients of this power series repeat.) Note that, for the first few coefficients, $E(\alpha^j) = R(\alpha^j)$ for $j = 1, 2, \ldots, \delta - 1$. So we can calculate these coefficients in terms of the received word $\boldsymbol{r}$. These are the only coefficients we will use.
$$\eta(X) = \sum_{j=1}^{\infty} E(\alpha^j)X^j = \sum_{j=1}^{\infty}\sum_{i\in\mathcal{E}} \alpha^{ij}X^j = \sum_{i\in\mathcal{E}}\sum_{j=1}^{\infty} \alpha^{ij}X^j = \sum_{i\in\mathcal{E}} \frac{\alpha^i X}{1-\alpha^i X}$$
Write this as $\dfrac{\omega(X)}{\sigma(X)}$ for the polynomial
$$\omega(X) = \sum_{i\in\mathcal{E}} \alpha^i X \prod_{j:j\neq i}(1 - \alpha^j X)\ .$$
Note that both $\sigma(X)$ and $\omega(X)$ have degree $|\mathcal{E}| \leqslant t$.

We have shown that $\sigma(X)\eta(X) = \omega(X)$. Writing this out explicitly and using $E(\alpha^k) = R(\alpha^k)$ for $k = 1, 2, \ldots, 2t$ we obtain
$$\left(\sigma_0 + \sigma_1 X + \ldots + \sigma_t X^t\right)\left(R(\alpha)X + R(\alpha^2)X^2 + \ldots + R(\alpha^{2t})X^{2t} + E(\alpha^{2t+1})X^{2t+1} + \ldots\right) =$$
$$= \omega_0 + \omega_1 X + \ldots + \omega_t X^t\ .$$
The coefficients of X^n for $t < n \leqslant 2t$ are
$$\sum_{j=0}^{t} \sigma_j R(\alpha^{n-j}) = 0$$
which do not involve any of the terms $E(\alpha^k)X^k$. So we get equations
$$\begin{pmatrix} R(\alpha^{t+1}) & R(\alpha^t) & \ldots & R(\alpha) \\ R(\alpha^{t+2}) & R(\alpha^{t+1}) & \ldots & R(\alpha^2) \\ \vdots & \vdots & & \vdots \\ R(\alpha^{2t}) & R(\alpha^{2t-1}) & \ldots & R(\alpha^t) \end{pmatrix} \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_0 \\ \vdots \\ \sigma_t \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}\ .$$
The matrix above is a $t \times (t+1)$ matrix, so there is always a vector $\sigma \neq \mathbf{0}$ in its kernel. This tells us what the error locator polynomial $\sigma(X)$ is and hence what the errors are that we should correct.

The special case of a BCH code where the field $\mathbb{F}$ is $\mathbb{F}_q$ and $N = q - 1$ is also called a Reed – Solomon code. These are very widely used. In particular, CDs are encoded using two interleaved Reed – Solomon codes. Both are defined over $\mathbb{F}_{2^8} = \mathbb{F}_{256}$ with $\delta = 5$. They have lengths $N = 32$ and 28 respectively. The interleaving spreads the information physically on the disc so that bursts of successive errors, as caused by a scratch, can be corrected more reliably. A CD can correct a burst of about 4,000 binary errors.

14: LINEAR FEEDBACK SHIFT REGISTERS

14.1 Definitions

Suppose that we have K registers each of which can contain one value from the finite field $\mathbb{F} = \mathbb{F}_q$ with q elements. Initially these contain the values $x_0, x_1, \ldots, x_{K-1}$, which is called the *initial fill.* At each time step, the values in the registers are shifted down and the final register is filled by a new value determined by the old values in all of the registers.

Denote the values in the registers at time $t = 0, 1, 2, \ldots$ by $X_0(t), X_1(t), \ldots, X_{K-1}(t)$. Then we have

$$\begin{aligned}
X_0(t+1) &= X_1(t) \\
X_1(t+1) &= X_2(t) \\
&\vdots \\
X_{K-2}(t+1) &= X_{K-1}(t) \\
X_{K-1}(t+1) &= f(X_0(t), X_1(t), \ldots, X_{K-1}(t)) \ .
\end{aligned}$$

The function f is called the *feedback function.* The system is called a *linear feedback shift register* (LFSR) when the feedback function is linear, say $f(X_0, X_1, \ldots, X_{K-1}) = -c_0X_0 - c_1X_1 - \ldots - c_{K-1}X_{K-1}$ so that

$$c_0X_0(t) + c_1X_1(t) + \ldots c_{K-1}X_{K-1}(t) + X_K(t) = 0$$

for each t.

Such a linear feedback shift register gives an infinite stream of values

$$X_0(0),\ X_0(1),\ X_0(2),\ ,\ldots,\ X_0(t),\ \ldots\ .$$

The values in register r are just these values with the first $r-1$ discarded and the others shifted back. This stream of values begins with the initial fill

$$X_0(0) = x_0,\ X_0(1) = x_1,\ X_0(2) = x_2,\ \ldots,\ X_0(K-1) = x_{K-1}$$

and satisfies the recurrence relation

$$c_0X_0(t) + c_1X_0(t+1) + c_2X_0(t+2) + \ldots c_{K-1}X_0(t+K-1) + X_0(t+K) = 0\ .$$

The polynomial $C(X) = c_0 + c_1X + c_2X^2 + \ldots + c_{K-1}X^{K-1} + X^K$ is the *feedback polynomial.* It is the characteristic polynomial for the recurrence relation and so determines the solutions.

It is simple to produce such feedback shift registers using computers and so to produce such streams. In superficial ways the stream of values appears random but we will see that this is not the case.

We will always assume that the first coefficient c_0 for the feedback function is not zero. For, if $c_0 = 0$, then the value in the 0th register does not affect any later values and we should consider the remaining $K-1$ registers as a simpler LFSR.

Proposition 14.1 Periodicity for LFSR
A linear feedback shift register is periodic.

This means that there is an integer N with $X_0(t+N) = X_0(t)$ for each time t. The smallest such integer $N > 0$ is the *period* for the LFSR.

Proof:

For each time t, let $\boldsymbol{V}(t)$ be the vector

$$\boldsymbol{V}(t) = \begin{pmatrix} X_0(t) \\ X_0(t+1) \\ X_0(t+2) \\ \vdots \\ X_0(t+K-1) \end{pmatrix} \in \mathbb{F}^K .$$

There are only a finite number q^K of vectors in $\mathbb{F}^K$ so the sequence $(\boldsymbol{V}(t))_{t\in\mathbb{N}}$ must eventually repeat a value taken earlier. Suppose that this first occurs at $\boldsymbol{V}(N)$ with $\boldsymbol{V}(N) = \boldsymbol{V}(j)$ for some $0 \leqslant j < N$.

The definition of the LFSR shows that

$$\boldsymbol{V}(t+1) = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -c_0 & -c_1 & -c_2 & \dots & -c_{K-1} \end{pmatrix} \boldsymbol{V}(t) .$$

Write this as $\boldsymbol{V}(t+1) = M\boldsymbol{V}(t)$. The matrix M has determinant $\pm c_0$, which we are assuming to be non-zero. So M is invertible.

We have $\boldsymbol{V}(t) = M^t\boldsymbol{V}(0)$, so the first repeat gives

$$M^N\boldsymbol{V}(0) = M^j\boldsymbol{V}(0) .$$

If j were not 0, then we could multiply by M^{-1} to get an earlier repeat. Hence we must have $j = 0$ and so $M^N\boldsymbol{V}(0) = \boldsymbol{V}(0)$. Consequently, the sequence of vectors $(\boldsymbol{V}(t))$ is periodic with period N. Looking at the first entry in these vectors shows that $(X_0(t))$ is also periodic. □

Exercise:
What happens when the coefficient c_0 is allowed to be 0? Do we still get periodicity when the feedback function f is not assumed to be linear? Is the matrix M in the proof periodic? What is its characteristic polynomial?

The matrix M in the proof above clearly maps $\mathbf{0}$ to itself. Hence, the largest the period can be is $N = q^K - 1$. In this case the sequence $\boldsymbol{V}(0), \boldsymbol{V}(1), \dots, \boldsymbol{V}(N-2), \boldsymbol{V}(N-1)$ takes every value in $\mathbb{F}^K \setminus \{0\}$ exactly once. In one period of length $q^K - 1$, we then see that 0 occurs $q^{K-1} - 1$ times while every other number in $\mathbb{F}$ occurs q^{K-1} times. In this, rather weak, sense the sequence is random. However, the sequence is periodic so it becomes entirely predictable once we have at least one period of data.

Let N be the period for a linear feedback shift register and set

$$\boldsymbol{W}(t) = \begin{pmatrix} X_0(t) \\ X_0(t+1) \\ X_0(t+2) \\ \vdots \\ X_0(t+N-1) \end{pmatrix} \in \mathbb{F}^N .$$

Then the vectors $\boldsymbol{W}(t+1)$ is a cyclic shift of $\boldsymbol{W}(t)$ because $X_0(t+N) = X_0(t)$. Also,

$$c_0\boldsymbol{W}(0) + c_1\boldsymbol{W}(1) + \dots + c_{K-1}\boldsymbol{W}(K-1) + \boldsymbol{W}(K) = \mathbf{0} .$$

So we see that the vectors $\boldsymbol{W}(0), \boldsymbol{W}(1), \dots, \boldsymbol{W}(K-1)$ span the code book for a cyclic code of length N.

14.2 The Berlekamp – Massey Algorithm

Suppose that you receive a sequence of values $(a_t)_{t\in\mathbb{N}}$ from the finite field $\mathbb{F}$ that you believe has come from a linear feedback shift register. Can you determine which linear feedback shift register has produced it?

This is simply a matter of solving linear equations to find the coefficients in the feedback polynomial. Suppose that the sequence came from a linear feedback shift register with K registers and feedback polynomial

$$C(X) = c_0 + c_1X + c_2X^2 + \ldots + c_{K-1}X^{K-1} + X^K \ .$$

Then we would have

$$c_0a_t + c_1a_{t+1} + \ldots + c_{K-1}a_{t+K-1} + a_{t+K} = 0$$

for every t. Consequently,

$$\begin{pmatrix} a_0 & a_1 & a_2 & \ldots & a_{K-2} & a_{K-1} \\ a_1 & a_2 & a_3 & \ldots & a_{K-1} & a_K \\ a_2 & a_3 & a_4 & \ldots & a_K & a_{K+1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{K-2} & a_{K-1} & a_K & \ldots & a_{2K-4} & a_{2K-3} \\ a_{K-1} & a_K & a_{K+1} & \ldots & a_{2K-3} & a_{2K-2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{K-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} . \qquad (*)$$

Hence the matrix above must have determinant 0.

Berlekmap and Massey formalised this as an algorithm to find the linear feedback shift register that generated (a_t). Begin with the smallest value of K that might be possible for the number of registers and compute the determinant

$$\begin{vmatrix} a_0 & a_1 & a_2 & \ldots & a_{K-2} & a_{K-1} \\ a_1 & a_2 & a_3 & \ldots & a_{K-1} & a_K \\ a_2 & a_3 & a_4 & \ldots & a_K & a_{K+1} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{K-2} & a_{K-1} & a_K & \ldots & a_{2K-4} & a_{2K-3} \\ a_{K-1} & a_K & a_{K+1} & \ldots & a_{2K-3} & a_{2K-2} \end{vmatrix} .$$

If this is non-zero, then we can not solve the problem using K registers. So increase K by 1 and repeat the process. If the determinant is 0, then solve $(*)$ above to find the coefficients c_j. Then set up a linear feedback shift register with initial fill $a_0, a_1, \ldots, a_{K-1}$ and feedback polynomial $C(X) = c_0 + c_1X + c_2X^2 + \ldots + c_{K-1}X^{K-1} + X^K$. check if the stream produced by this agrees with the original stream. If it does we are finished. If it does not, then consider other solutions (c_j) or increase K.

14.3 Power Series

Let $A(X) = a_0 + a_1X + \ldots + a_DX^D$ and $B(X) = 1 + b_1X + \ldots + b_KX^K$ be two polynomials in $\mathbb{F}[X]$. If we write $B(X) = 1 - \beta(X)$, then we have

$$\frac{1}{B(X)} = \frac{1}{1-\beta(X)} = \sum_{j=0}^{\infty} \beta(X)^j \ .$$

Expanding the powers of $\beta(X)$ gives a formal power series for $1/B(X)$. Multiplying this by $A(X)$ we obtain a formal power series for $A(X)/B(X)$, say

$$\frac{A(X)}{B(X)} = \sum_{j=0}^{\infty} u_jX^j \ .$$

In these formal power series we are not concerned about convergence but merely wish the coefficients of each power of X on each side of the equation to match. Hence we need

$$A(X) = B(X)\left(\sum_{j=0}^{\infty} u_j X^j\right)$$

which is equivalent to

$$a_n = \sum_{j=0}^{n} b_j u_{n-j} \qquad \text{for } n = 0, 1, 2, \dots \ .$$

Thus the sequence (u_j) satisfies

$$\begin{aligned} u_n &= a_n - \sum_{j=1}^{n} b_j u_{n-j} && \text{for } n = 0, 1, \dots, D; \\ u_n &= \quad\; - \sum_{j=1}^{n} b_j u_{n-j} && \text{for } n > D \ . \end{aligned}$$

This shows that the sequence (u_j) is the stream produced by a linear feedback shift register with feedback polynomial

$$C(X) = b_K + b_{K-1}X + b_{K-2}X^2 + \ldots + b_1 X^{K-1} + X^K \ .$$

This shows that determining whether a stream of numbers from $\mathbb{F}$ is the output of a LFSR is the same as determining whether a formal power series is the quotient of two polynomials. You should compare this to the method used to decode a BCH code in the previous lecture.

Exercise:
You learnt at school that a decimal repeats if and only if it represents a rational number. How does this relate to LFSRs and the periodicity proved in Proposition 14.1?

15: CRYPTOGRAPHY

15.1 Introduction

In many circumstances we want to send a message so that it can only be read by the intended recipient. In this case we talk about the message being *encrypted* or *enciphered*. This is an important and very actively studied area. As computers grow more powerful we need to devise ciphers that are more difficult to break.

We wish to transmit a message or *plaintext* which consists of a string of letters taken from a finite alphabet $\mathcal{A}$. Usually there are a variety of different methods to encode or *encipher* the message, each depending on a *key* taken from a finite set $\mathcal{K}$ of possible keys. We will write $e_k : \mathcal{A} \to \mathcal{B}$ for the encoding function or *encrypting function* corresponding to the key k. When the plain text is encrypted in this way we obtain the *ciphertext*. We must be able to decode or *decipher* the message, so there is a *decrypting function* $d_k : \mathcal{B} \to \mathcal{A}$ with $d_k(e_k(a)) = a$ for every letter a.

Example:
In a substitution cipher the key is a permutation κ of the alphabet $\mathcal{A}$. So each letter a is encrypted as $\kappa(a)$. This is deciphered by applying the inverse permutation.

For messages in English, where the alphabet is the usual Latin alphabet, it is generally easy to decipher such a code. For we look at the frequency of letters in English and compare this with the frequencies in the ciphertext. Using this, and the fact that the message should make sense, it is easy to find the permutation used and so decipher messages.

Example:
A more secure cipher is the Vigenère cipher, named after a French Diplomat, 1523. In this the key is a string $k_1k_2\dots k_D$ of letters from the alphabet $\mathcal{A}$. This string is repeated to give an infinite sequence

$$k_1k_2k_3\dots k_Dk_{D+1}k_{D+2}\dots k_{2D}k_{2D+1}k_{2D+2}\dots$$

with $k_{qD+r} = k_r$. Now a plaintext $a_1a_2\dots a_N$ is encrypted as $b_1b_2\dots b_N$ where

$$b_j \equiv a_j + k_j \pmod{|\mathcal{A}|} .$$

If the key is only one letter long, $D = 1$, then this is a particularly simple substitution cipher, called the Caesar cipher. When the key is longer, a given letter is encrypted differently depending on its place in the message. So simple frequency analysis will not work. Of course we could consider more complicated frequency analysis as was done by Babbage, who showed how to break the cipher provided we have access to a long enough ciphertext.

Babbage observed that common patterns of letters (words) will be enciphered in exactly the same way when the distance between their starting points is a multiple of D the length of the key. So we should look for reasonably long words that repeat in the ciphertext and then D must divide the distance between those repeats. ("The Code Book", by Simon Singh has a lot of similar historical information.)

What do mean by breaking a cipher? We will assume that our enemies know the methods we are using to encrypt messages but not the particular key we are using. So they know the possible functions e_k and d_k but not the value of k. If they intercept a piece of ciphertext, they will want to decode it or, even more damagingly, to work out the key so that they can decode any future ciphertexts.

There are various levels of attack we might protect against.

1. Ciphertext only attack
The enemies have a single piece of ciphertext.

2. Known Plaintext attack
The enemies have a piece of plaintext and the corresponding ciphertext.

3. Chosen Plaintext attack
The enemies can obtain arbitrary plaintexts of their choice and the corresponding ciphertexts.

A substitution cipher is vulnerable to an attack at level 1, provided that the ciphertext is reasonably long and the message is in English. The Vigenère cipher is also insecure at level 1, although a much longer ciphertext is required and the enemies need to work harder to break the cipher. In both cases the enemies will be able not simply to find the original message but to find the key and so decipher other messages. For modern ciphers, we would want them to be secure against a level 3 attack. So we need to devise better ciphers.

Note that every cipher will be vulnerable to a level 3 attack in some sense. For the system is finite: there are only finitely many letters and finitely many keys. So the enemies could carry out an exhaustive search. For each key $k \in \mathcal{K}$ they could encipher a fixed plaintext using that key and check if it agreed with the known ciphertext. To have a secure cipher, we want the amount of work involved in doing such an exhaustive search, or the time it would take, to be prohibitively large.

15.2 Equivocation

Suppose that a random plaintext message M is chosen from the set $\mathcal{M}$ of possible messages, with a certain probability distribution. Then we choose a random key $K \in \mathcal{K}$ independently of M. The ciphertext is then a random variable $C = c_K(M)$ in the set $\mathcal{C}$ of all possible ciphertexts.

We can think of this as transmitting the message M through a noisy channel to produce C. The noise comes from the choice of the key. The entropy $H(M|C)$ is the *message equivocation.* It measures the amount of uncertainty there is about the plaintext once we know the ciphertext. Similarly, the *key equivocation* $H(K|C)$ measures the amount of uncertainty about the key when we know the ciphertext. Unsurprisingly we have

Proposition 15.1 Message equivocation $\leqslant$ key equivocation
The message equivocation is no larger than the key equivocation.

Proof:
We have
$$H(K|C) = H(K,C) - H(C) = H(K,M,C) - H(C)$$
since $M = d_K(C)$ so M is a function of (K,C). Therefore,
$$\begin{aligned} H(K|C) &= H(K,M,C) - H(M,C) + H(M,C) - H(C) \\ &= H(K|M,C) + H(M|C) \end{aligned}$$
and so $H(K|C) \geqslant H(M|C)$. □

We say that a cipher has *perfect secrecy* if the ciphertext gives us no information about the plaintext, so $H(M|C) = H(M)$. This means that

$$H(M, C) = H(M) + H(C)$$

so M and C are independent. (Equivalently the mutual information $I(M, C) = H(C) - H(C|M) = H(C) + H(M) - H(M, C) = 0$.)

Proposition 15.2 Perfect secrecy
If a cipher has perfect secrecy, then there must be at least as many possible keys as there are possible plaintext messages.

Proof:
When we say "possible keys" or "possible messages" we mean that the probability is strictly positive.

Fix a message $m_o \in \mathcal{M}$ and a key $k_o \in \mathcal{K}$, both with strictly positive probability. Then $c_o = e_{k_o}(m_o)$ also has strictly positive probability. For any possible message $m \in \mathcal{M}$ we have

$$\mathbb{P}(C = c_o) = \mathbb{P}(C = c_o | M = m) \ .$$

So there must exist some key $k \in \mathcal{K}$ with $c_o = e_k(m)$. If two messages m_1, m_2 give the same key k, then $e_k(m_1) = c_o = e_k(m_2)$ and so $m_1 = m_2$. Hence the map $m \mapsto k$ is injective. □

This means that it is usually impractical to have perfect secrecy. However,there is a simple example that does give perfect secrecy. G. S. Vernam (1890 - 1960) and J. O. Mauborgne (1881 - 1971) proposed using a random key. We use a random number generator to produce a sequence

$$k_1 k_2 k_3 \dots$$

of letters from the alphabet $\mathcal{A}$. Each letter k_j is chosen uniformly from the $q = |\mathcal{A}|$ letters of the alphabet $\mathcal{A}$ and the choices are independent for each position j. Then a message $m = a_1 a_2 \dots a_N$ is enciphered as $b_1 b_2 \dots b_N$ where

$$b_j \equiv a_j + k_j \pmod{q} \ .$$

Each random letter k_j is only used once, so this cipher is called a *one-time pad.*

A one-time pad has perfect secrecy. For we have

$$\mathbb{P}(M = \boldsymbol{m}, C = \boldsymbol{c}) = \mathbb{P}(M = \boldsymbol{m}, K = \boldsymbol{c} - \boldsymbol{m}) = \mathbb{P}(M = \boldsymbol{m})\mathbb{P}(K = \boldsymbol{c} - \boldsymbol{m}) = \mathbb{P}(M = \boldsymbol{m})\frac{1}{q^N}$$

where the subtraction $\boldsymbol{c} - \boldsymbol{m}$ is done modulo q. Therefore, M and C are independent.

It is believed that a one-time pad is used to encrypt the Washington – Kremlin hotline. However, there are major problems with a one-time pad. First we need a way to produce genuinely random sequences of letters. This is not straightforward. Secondly, if the cipher is to be deciphered by the intended recipient, he or she needs to know the key sequence. So a key, just as long as the message, needs to be sent to the receiver in a secure way. In this way we have replaced the problem of transmitting a message securely by another — transmitting the key — of the same difficulty.

Various people have been tempted to replace the random key by a suitable pseudo-random sequence. For example we might use a linear feedback shift register to produce a sequence of supposedly random letters. This is vulnerable to a level 2 attack. For, if we know a piece of plaintext and the corresponding ciphertext, then we can compute the key sequence from their difference. Now the Berlekamp – Massey algorithm gives us a simple way to find the feedback polynomial and hence to find the key.

15.3 Unicity

If we try to break a simple substitution cipher by examining letter frequencies we are helped by the fact that we know the original message makes sense and that, for a reasonable length message, there is only one key that gives a sensible message from the ciphertext. We might ask how long a message we need for this argument to apply. This length is the idea behind the unicity.

Suppose that our message $m = a_1a_2a_3 \dots a_N$ is of length N. The letters are random variables A_j giving a random message $\boldsymbol{M} = A_1A_2A_3 \dots A_N$. We will assume that the entropy of this random sequence is NH for a constant H, the entropy per letter for our message. (Compare Lecture 5.) When this message is enciphered we get a ciphertext $\boldsymbol{C} = C_1C_2C_3 \dots C_N$. The *unicity* is the least value of N for which the conditional entropy $H(K|\boldsymbol{C})$ is 0.

Since K and $\boldsymbol{M}$ determine $\boldsymbol{C}$ and, conversely, K and $\boldsymbol{C}$ determine $\boldsymbol{M}$, we have

$$\begin{aligned} H(K|\boldsymbol{C}) &= H(K, \boldsymbol{C}) - H(\boldsymbol{C}) \\ &= H(\boldsymbol{M}, K, \boldsymbol{C}) - H(\boldsymbol{C}) \\ &= H(\boldsymbol{M}, K) - H(\boldsymbol{C}) \\ &= H(\boldsymbol{M}) + H(K) - H(\boldsymbol{C}) \end{aligned}$$

We are already assuming that $H(\boldsymbol{M}) = NH$. For most useful ciphers we have the ciphertext uniformly distributed, so $H(\boldsymbol{C}) = N \log_2 |\mathcal{C}|$. Finally, we choose the key K uniformly from $\mathcal{K}$, so $H(K) = \log_2 |\mathcal{K}|$. Therefore, the unicity N satisfies $0 = NH + \log_2 |\mathcal{K}| - N \log_2 |\mathcal{C}|$. So

$$N = \frac{\log_2 |\mathcal{K}|}{\log_2 |\mathcal{C}| - H} \ .$$

If we wish to be secure, we should not use a single key for more letters than the unicity.

Example:
For English a reasonable estimate of the entropy per letter is 1.2 bits. Suppose we use a substitution cipher where the key is a permutation of the 26 letters and the space. Then $|\mathcal{K}| = 27!$ so $\log_2 |\mathcal{K}| = 93.14$, $|\mathcal{C}| = 27$ and so the unicity is

$$N = \frac{\log_2 27!}{\log_2 27 - 1.2} = \frac{93.14}{4.75 - 1.2} = 26.2 \ .$$

So we expect to be able to find the key letter uniquely if the ciphertext is longer than 26 letters.

16: SYMMETRIC AND ASYMMETRIC CIPHERS

The ciphers that we have considered so far are *symmetric* in the sense that both the sender and the receiver need to know the key in order to encrypt or decrypt the message. Such ciphers are useful but require that the key is kept securely and can be securely communicated from the sender to the receiver. If there are many people I wish to communicate with, then I will need a different key for each and need to keep all these keys secure.

16.1 *Feistel Ciphers*

Suppose that we have some simple (non-linear) functions $f_k : \mathbb{F}_2^N \to \mathbb{F}_2^N$ for keys $k \in \mathcal{K}$. Then we can construct a more complicated and more secure cipher by using these functions in sequence.

Let $\boldsymbol{k} = (k_1, k_2, \dots, k_R)$ be a vector of R keys and consider a plaintext $(a_0, a_1) \in \mathbb{F}_2^N \times \mathbb{F}_2^N$. From these we construct a sequence of letters $a_0, a_1, a_2, \dots, a_{R-1}, a_R, a_{R+1}$ so that:

$$\begin{aligned}(a_0, a_1) \mapsto (a_1, a_0 + f_{k_1}(a_1)) = (a_1, a_2) \mapsto (a_2, a_1 + f_{k_2}(a_2)) = (a_2, a_3) \mapsto \dots \\ \dots \mapsto (a_R, a_{R-1} + f_{k_R}(a_R)) = (a_R, a_{R+1})\end{aligned}$$

Then the cipher text is (a_R, a_{R+1}). This is called a *Feistel cipher*.

Deciphering is done by using the keys in reverse order:

$$\begin{aligned}(a_R, a_{R+1}) \mapsto (a_{R+1} - f_{k_R}(a_R), a_R) = (a_{R-1}, a_R) \mapsto \dots \\ \dots \mapsto (a_3 - f_{k_2}(a_2), a_2) = (a_1, a_2) \mapsto (a_2 - f_{k_1}(a_1), a_1) = (a_0, a_1)\end{aligned}$$

and is equally easy to compute.

Many practical ciphers use this idea. For example, the DES (Data Encryption Standard, of the US National Bureaus of Standards, 1977 – 1988) uses Feistel ciphers. In this case $N = 32$ so we encipher binary strings of 64 bits. The transformation is repeated $R = 16$ times. Despite this complexity, it is possible to break such ciphers. Using a reasonably large parallel computing network it can be broken in a few days. Now more secure symmetric ciphers, such as the AES (Advanced Encryption Standard) are used.

16.2 Asymmetric Ciphers

We can also consider *asymmetric ciphers* where there are two keys, one for encrypting and the other for decrypting. For this to be sensible, it must be difficult to find the deciphering key from the enciphering one.

A particularly useful version of asymmetric ciphers is *public key cryptography.* Here, the receiver publishes for everyone to see the *public key* that should be used to encrypt messages intended for her. This enables anyone to compute the encrypting function and so to encrypt a message. Thus I can encrypt my plaintext message and send it to the receiver. She has a *private key* that enables her to decipher this encrypted message and so recover the plaintext. The private key is kept secure by the receiver, so that no one else can decipher the message.

For this idea to work, it must be hard to decipher a ciphertext without knowing the private key, even though we already know the public key. So, it should be simple to encrypt a message m as $c = e_k(m)$ using the function e_k that we know from the public key. However, if we are given the ciphertext c it should be very hard to find which plaintext m has $c = e_k(m)$. The function e_k is a "one-way function" where it is easy to compute $e_k(m)$ but difficult to find $e_k^{-1}(c)$.

Public key cryptography has the great advantage that we do not need to send the public key securely since it can be published for everyone to see. Also, the private key is kept securely by the receiver and does not need to be sent to anyone else. Even if we wish to use a symmetric cipher to exchange information, it may be useful to use a public key cryptogram to send the symmetric key securely.

The most commonly used public key cryptography relies on mathematical problems that are difficult to solve quickly. So the RSA cipher and the Rabin cipher rely on it being hard to factorise large integers. The ElGamal cipher relies on it being hard to find which power of a primitive root modulo N gives a particular integer. These problems have been found to be very difficult to solve, so we hope that breaking the ciphers is equally difficult. Of course, if there is a breakthrough and someone discovers a very fast method to solve these problems, then the ciphers will immediately become insecure. Even without this, as computers become faster the size of integers that can be factorised in a reasonable time grows larger, so we need to move to more complicated ciphers of the same type.

Before we look at examples of public key cryptography we need to think about what makes a computation hard or lengthy.

17: COMPLEXITY

All of the problems that we are considering are finite and so could be solved simply by considering all of the possibilities. However, some are much easier to solve than others. Consider for example:

"Factorise $20,989$" compared with "Show that $139 \times 151 = 20,989$."

or

"Find n with $2^n \equiv 5 \pmod{211}$" compared with "Verify that "$2^{132} \equiv 5 \pmod{211}$".

In each case it takes a long time to do the first and is quick to check the second. We will consider this relationship by asking how long it takes to perform certain calculations.

17.1 Polynomial Time

How long will it take to compute the value $f(n)$ of some function f? This is likely to depend on the value of n and, in particular, on how big it is. We will suppose that n is written in binary and has B digits (so $n < 2^B$). Each elementary operation, such as adding two binary bits, or comparing two bits, takes a maximum time τ. So the time to compute $f(n)$ will be at most the time τ multiplied by the number of elementary operations required to do the computation. We will say that f can be *computed in polynomial time* if there is an algorithm to compute $f(n)$ that takes at most time cB^k for all natural numbers n with B bits. Here c and k are some constants independent of n. Similarly, a function of several variables $f(n_1, n_2, \dots, n_r)$ can be computed in polynomial time if there is an algorithm that computes $f(n_1, n_2, \dots, n_r)$ in at most time cB^k where $B = B_1 + B_2 + \dots + B_r$ is the sum of the bit lengths of the numbers $n_1, n_2, \dots, n_r$.

It is easy to give examples of functions that can be computed in polynomial time.

(a) Adding or subtracting binary integers.

(b) Multiplying binary integers.

(c) Division of binary integers to give a quotient and remainder.

(d) Computing highest common factors, using Euclid's algorithm.

(e) Modular arithmetic (addition, subtraction, multiplication, division) modulo N.

(f) Exponentiation modulo N. To compute α^n, expand n in binary as $\sum n_j 2^j$ with each $n_j = 0$ or 1. Then compute the powers $\alpha, \alpha^2, \alpha^4, \dots, \alpha^{2^j}, \dots, \alpha^{2^B}$ by repeated squaring and

$$\alpha^n = \prod_{n_j=1} \alpha^{2^j}$$

as a product.

(g) Testing primality. Agrawal, Kayal and Saxena (2002) gave a polynomial time algorithm to test primality.

However, there are functions for which polynomial time algorithms are not known. For example:

1. **Factoring**
   An integer N is known to be the product of two primes p and q. Find these primes.

2. **Discrete Logarithm**
   Let α be a primitive root modulo the prime p., so α is a generator of the cyclic group $\mathbb{F}_p^\times$. Given $x \in \mathbb{F}_p^\times$, find n with $\alpha^n \equiv x \pmod{p}$. We call n the *logarithm of x to base α* and denote it by $\log_\alpha x$.

The elementary methods for computing 1 and 2 take much longer than polynomial time. To factor N we could try dividing by successive primes up to $N^{1/2}$. This takes time $O(N^{1/2}) = O(2^{B/2})$. To compute the discrete logarithm $\log_\alpha x$, set $m = \lceil p^{1/2} \rceil$ and write $n = qm + r$ with $0 \leqslant q, r < m$. Then

$$\alpha^n \equiv x \pmod{p} \qquad \Leftrightarrow \qquad (\alpha^m)^q \equiv x\alpha^{-r} \pmod{p} \ .$$

So compute $(\alpha^m)^q$ and $x\alpha^{-r}$ for all $0 \leqslant q, r < m$ and see where they agree. This takes time $O(p^{1/2}\log p) = O(2^{B/2}B)$.

Significantly better methods are known but these are still well short of polynomial time. Using number field sieves we can achieve a time

$$O\left(\exp(cB^{1/3}(\log B)^{2/3})\right) \ .$$

In practice, it seems to take a long time to solve either 1 or 2 for random large values of p, q and x. We will describe various ciphers based on the difficulty of solving these problems.

The RSA cipher relies on factoring being difficult. Up until 2007 the RSA laboratories offered rewards for factoring large challenge numbers. The RSA-B challenge involves factoring a number with B binary bits. Although the prizes have now ceased, this is still used as a test of computing power. The recent successes were

Challenge Number	Decimal digits	Factored	Prize
RSA-576	174	December, 2003	\$10,000
RSA-640	193	November, 2005	\$20,000
RSA-704	212	July, 2012	(\$30,000)
RSA-768	232	December, 2009	(\$50,000)
RSA-896	270		(\$75,000)
RSA-1024	309		(\$100,000)

(See *http://www.emc.com/emc-plus/rsa-labs/historical/ the-rsa-factoring-challenge-faq.htm* or *http://en.wikipedia.or*
.)

17.2 Modular Arithmetic

For each natural number N, the integers modulo N will be denoted by $\mathbb{Z}_N$. The set of integers modulo N that are coprime to N:

$$\mathbb{Z}_N^\times = \{a = 0, 1, 2, \dots, N-1 : (a, N) = 1\}$$

is a multiplicative group of order $\varphi(N)$ — the *Euler totient function.* Hence Lagrange's theorem shows that

$$a^{\varphi(N)} \equiv 1 \pmod{N} \qquad \text{for each } a \text{ with} \qquad (a, N) = 1 \ .$$

This is the *Euler – Fermat theorem.*

When N is a prime p we know that $\varphi(p) = p - 1$, and $\mathbb{Z}_p^\times = \mathbb{F}_p^\times$ consists of all the non-zero elements in the field $\mathbb{F}_p$. The set $\mathbb{Z}_p^\times = \mathbb{F}_p^\times$ is a cyclic group of order $\varphi(p) = p - 1$. An integer a is a *primitive root modulo* p if a is a generator of this cyclic group. The Euler – Fermat theorem gives Fermat's little theorem:

$$a^p \equiv a \pmod{p} \qquad \text{for all integers } a \ .$$

When N is a product of two distinct primes, say $N = pq$, then $\varphi(N) = (p-1)(q-1)$. Fermat's little theorem shows that, for each integer k,

$$a^{k(p-1)+1} \equiv a \pmod{p} \qquad \text{for all intgers } a\ .$$

So, in particular, $a^{k\varphi(N)+1} \equiv a \pmod{p}$. Similarly, $a^{k\varphi(N)+1} \equiv a \pmod{q}$. So we obtain

$$a^{k\varphi(N)+1} \equiv a \pmod{N} \qquad \text{for all integers } a\ .$$

We will also need:

Theorem 17.1 Chinese Remainder Theorem
Let p, q be two coprime integers, so there are integers a, b with $ap + bq = 1$.
The equations $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$ are equivalent to

$$x \equiv c + ap(d-c) \pmod{pq}\ .$$

Proof:
If $x = c + ap(d-c) = c + (1-bq)(d-c) = d - bq(d-c)$, then it is clear that $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$.

Conversely, suppose that $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$, so $p|(x-c)$ and $q|(x-d)$. Then

$$p|(x - c - ap(d-c)) \qquad \text{and} \qquad q|(x - d - bq(d-c))\ .$$

This shows that pq divides $x - d - bq(d-c) = x - c - ap(d-c)$, so

$$x \equiv c + ap(d-c) \pmod{pq}\ .$$

□

The Chinese Remainder theorem shows that the multiplicative group homomorphism

$$\mathbb{Z}_{pq}^{\times} \to \mathbb{Z}_p^{\times} \times \mathbb{Z}_q^{\times}\ ; \quad a \mapsto (a \pmod{p},\ a \pmod{q})$$

is an isomorphism. In particular, $\varphi(pq) = \varphi(p)\varphi(q)$ for coprime integers p and q.

Proposition 17.2 Quadratic residues
Let p be an odd prime. In the field $\mathbb{F}_p$, the only square root of 0 is 0; precisely $\frac{1}{2}(p-1)$ elements have no square root and the remaining $\frac{1}{2}(p-1)$ have exactly two square roots.

Proof:
Since $\mathbb{F}_p$ is a field, we have

$$x^2 \equiv y^2 \pmod{p} \qquad \Leftrightarrow \qquad (x-y)(x+y) \equiv 0 \pmod{p}\ .$$

Hence, the squaring map $s : \mathbb{F}_p \to \mathbb{F}_p$; $x \mapsto x^2 \pmod{p}$ has $s(x) = s(y)$ if and only if $y = \pm x$. This means that $s^{-1}(0) = \{0\}$ and every other square has two square roots. Hence there must be $\frac{1}{2}(p-1)$ squares. □

Proposition 17.3 Square roots modulo pq
Let N be the product of two distinct, odd primes p and q. Suppose that the integer a is a square modulo N. Then either:

(a) $(a, N) = 1$ *and there are exactly* 4 *square roots of* a *modulo* N.

(b) $(a, N) = p$ *or* q *and there are exactly* 2 *square roots of* a *modulo* N.

(c) $a \equiv 0 \pmod{N}$ *and* 0 *is the only square root of* 0 *modulo* N.

Proof:

For an integer x we have

$$x^2 \equiv a \pmod{N} \qquad \Leftrightarrow \qquad x^2 \equiv a \pmod{p} \qquad \text{and} \qquad x^2 \equiv a \pmod{q} \ .$$

When $(a, N) = 1$, we have 2 solutions to $x^2 \equiv a \pmod{p}$. Similarly, there are 2 solutions to $x^2 = a \pmod{q}$. Now the Chinese remainder theorem shows that there are 4 solutions to $x^2 \equiv a \pmod{N}$.

When $p|a$ there is only one solution to $x^2 \equiv a \pmod{p}$. This gives parts (b) and (c). $\square$

18: PUBLIC KEY CIPHERS

18.1 RSA Ciphers

The Rivest – Shamir – Adelman (RSA) ciphers are our first example of public key ciphers. They are very widely used and rely on the difficulty in factoring products $N = pq$ of two large primes.

To make an RSA cipher, I first choose two large primes p, q and set $N = pq$. (Typically these are primes with at least 800 binary bits.) Then I choose an exponent e randomly with e coprime to $\varphi(N) = (p-1)(q-1)$. I can always ensure that $(e, \varphi(N)) = 1$ by taking e as a prime number larger than both p and q. This number e is called the *encrypting exponent*. Euclid's algorithm allows me to find integers d, k with

$$de - k\varphi(N) = 1$$

(in polynomial time). The integer d is called the *decrypting exponent*.

The public key will be (N, e). The encrypting function is

$$a \mapsto a^e \pmod{N} .$$

The private key will be d. The decrypting function is

$$c \mapsto c^d \pmod{N} .$$

The Euler – Fermat theorem shows that

$$(a^e)^d \equiv a^{de} \equiv a^{k\varphi(N)+1} \equiv a \pmod{N} .$$

So the decrypting function is inverse to the encrypting function.

An enemy who intercepts a ciphertext needs to find the plaintext knowing only the public key. This appears to be tantamount to finding the factors p and q of N so that $\varphi(N)$ can be computed.

Theorem 18.1 Security of the RSA ciphers
Suppose that we have an algorithm to determine the private key for an RSA cipher when we are given the public key. Then the algorithm permits us to factorise products of two distinct primes, with probability arbitrarily close to 1.

If we can do this in polynomial time, then we have a very effective way to break RSA ciphers but the theorem shows that this would also give us an equally effective way to factorise. No such algorithm is known.

Proof:
We are assuming that we have an algorithm that gives the decrypting exponent d in terms of N and e. Then $a^{de} \equiv a \pmod{N}$ for every $a \in \mathbb{Z}_N$.

Write $de - 1 = 2^s r$ for some odd integer r. Let $\mathrm{ord}_p(x)$ denote the order of an element x in the group $\mathbb{Z}_p^\times$. Set $X = \{x \in \mathbb{Z}_N^\times : \mathrm{ord}_p(x^r) \neq \mathrm{ord}_q(x^r)\}$. We will prove various properties of X that eventually prove our theorem.

Lemma A

If $x \in X$, then we can factorise N.

Proof:
If $x \in X$, then $y = x^r$ satisfies $y^{2^s} = x^{2^s r} = x^{de-1} = 1$ in $\mathbb{Z}_N^\times$, so $\mathrm{ord}_p(y)$ and $\mathrm{ord}_q(y)$ must be powers of 2. Suppose that $\mathrm{ord}_p(y) = 2^t < \mathrm{ord}_q(y)$. Then $y^{2^t} \equiv 1 \pmod{p}$ but $y^{2^t} \not\equiv 1 \pmod{q}$. So

$$(y^{2^t} - 1, N) = p .$$

We could therefore use Euclid's algorithm to find one of the factors of N.

Thus, to factorise N when we have $x \in X$, we compute $(y^{2^t} - 1, N)$ for $t = 0, 1, 2, \dots, s$. We know that it must be p for one of these choices and so we obtain a factor of N. $\square$

Now we wish to count how may elements there are in X.

Lemma B

For the prime p and the exponent r as above we have

$$|\{x \in \mathbb{Z}_p^\times : \mathrm{ord}_p(x^r) = c\}| \leqslant \tfrac{1}{2}(p-1)$$

for every possible order c.

Proof:

Let α be a primitive root modulo p. So $\mathbb{Z}_p^\times$ is the cyclic group generated by α. Then $\alpha^{2^s r} \equiv 1 \pmod{N}$ so we certainly have $\mathrm{ord}_p(\alpha^r)|2^s$. Let $\mathrm{ord}_p(\alpha^r) = 2^t$ for some $0 \leqslant t \leqslant s$.

If $x = \alpha^k$, then $x^r = (\alpha^r)^k$ and so

$$\mathrm{ord}_p(x^r) = \frac{2^t}{(2^t, k)} \ .$$

Hence, $\mathrm{ord}_p(x^r) = 2^t$ if and only if k is odd. There are $\frac{1}{2}(p-1)$ such values for x in $\mathbb{Z}_p^\times$. The remaining $\frac{1}{2}(p-1)$ values for x have $\mathrm{ord}_p(x^r) < 2^t$. So no more than $\frac{1}{2}(p-1)$ elements from $\mathbb{Z}_p^\times$ can have $\mathrm{ord}_p(x^r) = c$. □

Lemma C

$|X| \geqslant \frac{1}{2}(p-1)(q-1) = \frac{1}{2}\varphi(N)$.

Proof:

The Chinese remainder theorem Theorem 17.1 shows that $|X|$ is

$$|\{(x,y) \in \mathbb{Z}_p^\times \times \mathbb{Z}_q^\times : \mathrm{ord}_p(x^r) \neq \mathrm{ord}_q(y^r)\}| \ .$$

For each $y \in \mathbb{Z}_q^\times$ we have shown that

$$\{x \in \mathbb{Z}_p^\times : \mathrm{ord}_p(x^r) \neq \mathrm{ord}_q(y^r)\}$$

has at least $\frac{1}{2}(p-1)$ elements. Therefore $|X| \geqslant \frac{1}{2}(p-1)(q-1) = \frac{1}{2}\varphi(N)$. □

Proof of Theorem 18.1 (continued)
Choose an integer x randomly from $\mathbb{Z}_N^\times$. The probability that $x \in X$ is at least $\frac{1}{2}$. When $x \in X$ we know how to find a factor of N. When $x \notin X$, choose another random value for x. After k such random choices we will have found a factor of N with probability at least $1 - \left(\frac{1}{2}\right)^k$.

□

Note that the theorem shows that finding the private key is as hard as factoring N. There might well be other ways to decipher a particular ciphertext without finding the private key d. Rivest, Shamir and Adelman conjectured that any algorithm that allows us to decipher messages would also allow us to factorise N. However, this has not been proved.

18.2 Rabin Ciphers

The Rabin cipher is another public key cipher that relies on the difficulty of factoring products $N = pq$ of two large primes. For this we need to consider finding square roots modulo a prime.

Lemma 18.2
Let p be a prime of the form $4k-1$. If $c \equiv a^2 \pmod{p}$ then $a \equiv \pm c^k \pmod{p}$.

Proof:

If $c \equiv a^2 \pmod{p}$, then Fermat's little theorem gives

$$c^{2k} \equiv a^{4k} \equiv a^{(p-1)+2} \equiv a^2 \pmod{p} .$$

So $c^k \equiv \pm a \pmod{p}$. □

To make a Rabin cipher I first choose two large primes p, q of the form $p = 4k - 1$ and $q = 4m - 1$. Set $N = pq$. Then I will create a cipher for the alphabet $\mathbb{Z}_N^\times$.

The public key will be N and the encrypting function is

$$a \mapsto a^2 \pmod{N} .$$

(Usually we restrict the alphabet so that $(a, N) = 1$ and $a > N^{1/2}$.) The private key will be (p, q).

Suppose that we have received a ciphertext $c \equiv a^2 \pmod{N}$ and know the private key. Then Lemma 18.2 shows that

$$a \equiv \delta c^k \pmod{p} \qquad \text{and} \qquad a \equiv \varepsilon c^m \pmod{q}$$

where δ, ε are each ± 1. Find integers u, v with $up + vq = 1$. Then the Chinese remainder theorem shows that

$$a \equiv \delta c^k + up(\varepsilon c^m - \delta c^k) \pmod{N} .$$

All four of these possible values can occur and are distinct by Proposition 17.3 . To decipher c we find all four square roots modulo N and choose the one that makes sense. Our messages should contain enough redundancy for only one of the four choices to make sense.

When we do not know the private key, breaking the Rabin cipher is as hard as factorising N.

Theorem 18.3 Security of Rabin ciphers
An algorithm to decipher the Rabin cipher gives an algorithm to factorise N.

Proof:

An algorithm to decipher the Rabin cipher must give one of the four square roots of a ciphertext c modulo N which we obtain as $c \equiv x^2 \pmod{N}$.

Choose $a \in \mathbb{Z}_N^\times$ at random. This algorithm gives a particular square root x of $c \equiv a^2 \pmod{N}$, so $x^2 \equiv a^2 \pmod{N}$. There are 4 distinct choices for a that give the same value for c. Two of these give $x \equiv \pm a$ but the other two do not. For these other two we must have

$$x^2 - a^2 \equiv (x + a)(x - a) \equiv 0 \pmod{N}$$

but neither $x + a$ nor $x - a$ is divisible by N. Therefore, $(N, x - a) \neq 1$.

This means that, with probability $\frac{1}{2}$, we find a non-trivial factor $(N, x - a)$ of N. If we repeat this r times, the chance of finding a factor of N is at least $1 - \left(\frac{1}{2}\right)^r$. □

19: DISCRETE LOGARITHM CIPHERS

In this lecture we will look at some ciphers founded on the difficulty of solving the discrete logarithm problem. Throughout, p will be a large prime number and γ will be a primitive root modulo p. So all of the non-zero elements of $\mathbb{Z}_p$ are powers of γ. We will assume that the values of p and γ are published and known to everyone.

19.1 Diffie – Hellman Key Exchange

First we look at how to establish a common secret key so two people can use a symmetric cipher to communicate securely. The method described is the *Diffie – Hellman key exchange.*

Let Alice and Bob be the two people. They wish to agree on a key $k \in \mathbb{Z}_p$ to use. Alice chooses a random $a \in \mathbb{Z}_{p-1}$; computes $A = \gamma^a$; and then publishes A. Similarly, Bob chooses a random $b \in \mathbb{Z}_{p-1}$; computes $B = \gamma^b$; and publishes B. Then they take the key to be

$$k \equiv B^a \equiv A^b \pmod{p} .$$

Alice knows the value of a and the published number B so she can compute B^a. Bob knows b and the published number A so he can compute A^b. However,

$$B^a \equiv (\gamma^b)^a = (\gamma^a)^b \equiv A^b \pmod{p}$$

so Alice and Bob may take the common value of B^a and A^b as their common key.

If an enemy can compute discrete logarithms efficiently, then he can find $a = \log_\gamma A$ from the published values of p, γ and A. Then he can compute the key as B^a. Diffie and Hellman conjectured but did not prove that finding the value of this key from the published values A and B is equivalent to the discrete logarithm problem.

Shamir showed how we can use this idea to communicate securely without any public keys. Alice and Bob take $\mathbb{Z}_p$ as their alphabet. As in the Diffie – Hellman key exchange, Alice chooses a random $a \in \mathbb{Z}_{p-1}$ and computes $A = \gamma^a$. Alice also finds an integer a' with $aa' \equiv 1 \pmod{p-1}$ by using Euclid's algorithm. Alice keeps both numbers a and a' securely. Similarly, Bob chooses a random $b \in \mathbb{Z}_{p-1}$; computes $B = \gamma^b$; and finds an integer b' with $bb' \equiv 1 \pmod{p-1}$. He keeps both b and b' securely.

To send a message $m \in \mathbb{Z}_p$ from Alice to Bob, Alice computes $c \equiv m^a \pmod{p}$ and sends c to Bob. Then Bob computes $d \equiv c^b \pmod{p}$ and sends d back to Alice. Alice now computes $e \equiv d^{a'} \pmod{p}$ and sends e to Bob. Finally, Bob computes $e^{b'} \pmod{p}$. Fermat's little theorem shows that

$$e^{b'} \equiv d^{a'b'} \equiv c^{ba'b'} \equiv c^{a'} \equiv m^{aa'} \equiv m \pmod{p} .$$

So Bob has recovered the plaintext m.

	Alice	Bob
Private	a	b
Private	a' with $aa' \equiv 1 \pmod{p-1}$	b' with $bb' \equiv 1 \pmod{p-1}$

$$\begin{tikzcd}
m \arrow[d] & & \\
c = m^a \pmod{p} & \longrightarrow & c \arrow[d] \\
d \arrow[d] & \longleftarrow & d \equiv c^b \pmod{p} \\
e \equiv d^{a'} \pmod{p} & \longrightarrow & e \arrow[d] \\
& & f \equiv e^{b'} \pmod{p}
\end{tikzcd}$$

This method has the advantage that no keys have to be published in order for Alice and Bob to communicate. However, it takes three times as long to transmit a message.

19.2 The Elgamal Cipher

Elgamal showed how to adapt the Diffie – Hellman key exchange to give a cipher.

Bob wishes to send encrypted messages to Alice. So Alice chooses a random private key $a \in \mathbb{Z}_{p-1}$; computes $A \equiv \gamma^a \pmod{p}$; and publishes A as her public key. To send a message $m \in \mathbb{Z}_p$, Bob first chooses a random number $b \in \mathbb{Z}_{p-1}$ and sends the pair

$$(c_0, c_1) = (\gamma^b, A^b m) \in \mathbb{Z}_p \times \mathbb{Z}_p \ .$$

Alice can decipher this by computing $c_1 c_0^{-a} \pmod{p}$. For we have

$$c_o^a \equiv (\gamma^b)^a \equiv \gamma^{ab} \equiv (\gamma^a)^b \equiv A^b \pmod{p} \ .$$

So $c_1 c_0^{-a} \equiv m \pmod{p}$.

If an enemy knows one plaintext m and the corresponding ciphertext (c_0, c_1), then he can find the two public keys $A \equiv \gamma^a \pmod{p}$ and $B \equiv \gamma^b \pmod{p}$ used in the Diffie – Hellman key exchange. Breaking the Elgamal cipher is equivalent to breaking the Diffie – Hellman key exchange. We hope that both are as hard as computing discrete logarithms.

20: SIGNATURES

So far we have been concerned with how to send messages securely from one person to another. By encrypting the message we try to ensure that, if the message is intercepted, no-one except the intended recipient can decipher it. However, there are many other security concerns. When we receive a message we might want to know who sent it to protect us from forgeries. We might need to know that it has not been altered. We might need to prove that it came from a particular person. For all of these we can adapt the ciphers we have studied to give us the required reassurance.

20.1 Guarantees

Bob sends Alice a message. They may both be concerned about:

Secrecy
No third party can read their message. The ciphers we described in the past two lectures achieve this.

Integrity
No third party has altered the message. For example, if Bob is the customer of the bank owned by Alice, and sends an encrypted instruction to pay £100 to Charles, then Alice needs to be sure that the amount and recipient have not been altered.

The ciphers we have used do not automatically achieve this. For example, suppose that the message to the bank consisted of two numbers (a, m) the first specifying the recipient and the second being the number of pounds to pay them. These are encrypted using the RSA algorithm as $(c_0, c_1) = (a^e, m^e)$. If this message is intercepted, it may be impossible to decipher but it can still be sent again requesting further payments. Even more worryingly, the message could be altered to (c_0, c_1^3) and then sent. This last is called a *homomorphism attack*. It uses knowledge of the form of the cipher.

Authenticity
Alice can be sure that Bob sent the message. The usual signatures at the bottom of letters are intended to ensure that the recipient knows who sent it. We want to develop similar signatures for encrypted messages.

For example, if an enemy intercepts messages that end with a signature identifying the sender, he can swap the signatures to create chaos even if he can not understand the messages. We would like a signature that guarantees that the particular message comes from an identified sender.

Non-repudiation
Alice can prove that Bob sent a particular message. This is the role played by signatures on legal documents. A court can be convinced that a particular person signed the document and so accepted its contents. If electronic communications are to be used in law they require a similar guarantee.

All of these problems are interrelated and we may well require a combination of the guarantees described above.

20.2 Hash Functions

Suppose that we have a function $h : \mathcal{A}^* \to \mathcal{B}$ from all messages using the alphabet $\mathcal{A}$ into another alphabet $\mathcal{B}$. There are infinitely many messages but $\mathcal{B}$ is finite, so there are necessarily many messages with the same h-values Nonetheless, it should be hard to find two messages with the same h-value — a *collision*. If it is simple to compute $h(m)$ for any message m but hard to find a collision, then h is called a *hash function*.

Hash functions are similar to ciphers but they compress the information. They reduce any message to a letter in a fixed alphabet. If the message is changed then it is very likely that the hash value will also change. So we can use the hash value to test the integrity of a message. In this sense it acts as a digest or summary of the message.

Example:
The md5sum of a computer file is a 128 bit binary sequence computed for a file. The value is usually represented as a hexadecimal string of length 32. For instance the md5sum of a text file containing the sentence "The quick brown fox jumped over the lazy dog's back." is

```
ed6761a9a0d26cbe6c7a9666e07bf08d .
```

Changes to the file will, with very high probability change the md5sum. For example, if we change the capital letter to a small letter we get:

```
9c9d6b537e1ccc8623ee953bb442d147
```

which is very different.

The md5sum is used to check the integrity of computer files. If I download a large file over the internet, it may be corrupted either accidentally or maliciously. So I compute its md5sum and compare it with the md5sum published by the provider of the file.

It is often simple to produce hash functions from ciphers. Consider, for example, the RSA cipher with public key (N, e) and private key d. For any letter a I can compute $a^e \pmod{N}$ and then apply a permutation to the binary bits of a^e to give $\sigma(a^e)$. For a message $m = a_1a_2a_3 \dots a_K$ I compute $\sigma_k(a_k^e)$ for a sequence of different permutations σ_k, and then add the results modulo N. Changes to letters have an effect on the hash value that is difficult to predict so it appears to be difficult to find collisions.

In practice hash functions tend to be produced by less algebraic means. A message is first broken up into blocks of fixed length M, say $B_1, B_2, B_3, \dots, B_K$, padding the last block if necessary. Then we choose a compression function (or a sequence of compression functions) $C : \mathcal{A}^M \times \mathcal{A}^M \to \mathcal{A}^M$. Starting with an fixed initial block J, we compute successively:

$$C_1 = J,\ C_2 = C(C_1, B_1),\ C_3 = C(C_2, B_2), \dots,\ C_{n+1} = C(C_n, B_n),\ \dots,\ C_{K+1} = C(C_K, B_K)\ .$$

The final value C_{K+1} when we have processed the entire message is the hash value.

We will see that hash functions provide a convenient way to guarantee messages.

20.3 RSA Signatures

Bob wishes to sign a message to show that he sent it. To do this he can use his public key cipher. Suppose that e_B is his encrypting function, which is published for everyone, and d_B his decrypting function, which he keeps private. For simplicity we will suppose that we are dealing with a cipher where the encrypting function and the decrypting function are inverses of each other. So $e_B \circ d_B = I$ as well as $d_B \circ e_B = I$. This is true, for example, for the RSA cipher.

For a simple signature, Bob just takes his name n, computes $s = d_B(n)$ using his private decrypting function, and appends s to his message. Anyone receiving this signature can calculate $e_B(s)$ using Bob's public encrypting function and this is Bob's name $e_B(d_B(n)) = n$. No one else has access to Bob's private decrypting function, so they would find it very hard to sign the message in this way. Note, however, that there is nothing to prevent an enemy from separating Bob's signature from a message he intercepts and attaching it to others spuriously. To avoid this we need a signature that guarantees the integrity of the message.

Bob wishes to send the message m. He adds to this a signature $s = d_B(h(m))$ where h is a publicly known hash function. So he sends (m, s).

When Alice receives a message (m, s) she can confirm that Bob sent it by using his public key to compute $e_B(s)$. Then she checks that $h(m) = e_B(s)$. Only Bob has the decrypting function d_B so only he could have signed the message with a signature satisfying $e_B(s) = h(m)$. Alice can also be confident that the message has not been altered since any alteration to m would require a corresponding alteration to $d_B(h(m))$.

Bob's signature can also be used to prove to a third party that the message was indeed from Bob. For, if we believe that the cipher we are using is secure, then no one except Bob could have sent a signed message (m, s) with $e_B(s) = h(m)$ without knowing Bob's private decrypting function d_B. Note, however, that Bob can invalidate this by telling others his private key and so allowing them to forge his signature. Furthermore, Alice can not alter the message m that she received without discovering how to make the corresponding change to $d_B(h(m))$.

The message in this example is not encrypted but, if it needs to be, we could use Alice's public key and send $(e_A(m), d_B(e_A(m)))$.

The hash function plays two important roles in this signature scheme. It reduces the length of the signature but it also makes it hard to forge. Suppose, for example, that Bob simply signed the message m with $s = d_B(m)$ and sent $(m, d_B(m))$. It is still possible for Alice to verify that the message is unaltered and came from Bob. However, any enemy can send $(e_B(c), c)$ using Bob's public key and these also appear to be correctly signed by Bob. However, the enemy has no way of knowing what the "message" $e_B(c)$ says or, indeed, if it makes any sense. (Such forged messages are known as *existential forgeries.*) The hash functions defeats this sort of forgery.

We also need to be confident that the message was signed and sent at the correct time, so we should insist that all messages are time stamped. This prevents messages being resent as if they were new.

20.4 The Elgamel Signature Scheme

The Elgamal cipher can also be used to sign messages. Let p be a large prime and γ a primitive root modulo p. Bob chooses $b \in \mathbb{Z}_{p-1}$ and publishes his public key $B = \gamma^b \in \mathbb{Z}_p^\times$. Let h be a hash function that takes values in $\mathbb{Z}_{p-1}$. To send a message m Bob chooses a random exponent k coprime to $p-1$ and puts

$$r \equiv \gamma^k \in \mathbb{Z}_p^\times \qquad s \equiv \frac{h(m) - br}{k} \pmod{p-1} \ .$$

(Since $(k, p-1) = 1$, we know that k is invertible modulo $p-1$.) Bob's signature is (r, s) so he sends the message (m, r, s) to Alice.

When Alice receives a triple $(m, r.s)$ she checks whether $\gamma^{h(m)} \equiv B^r r^s \pmod{p}$. If the triple is the message (m, r, s) from Bob then we have

$$\gamma^{h(m)} \equiv \gamma^{br+ks} = (\gamma^b)^r(\gamma^k)^s \equiv B^r r^s \pmod{p}$$

so Alice's check succeeds. It is believed that the only way to forge this signature is to solve the discrete logarithm problem.

In the Elgamal signature scheme it is vital that a different random exponent k is chosen for each message. For suppose that two messages m_1, m_2 are signed using the same value for k, giving (m_1, r, s_1) and (m_2, r, s_2). Then

$$h(m_1) - h(m_2) = k(s_1 - s_2) \pmod{p-1} .$$

Recall that $xs \equiv h \pmod{m}$ has either no solutions for x or else (m, s) solutions modulo m. Hence, there are $(p-1, s_1 - s_2)$ solutions for k modulo $p-1$. Choose the one that gives the correct value for $r \equiv \gamma^k \pmod{p}$. Then $s_1 \equiv \dfrac{h(m_1) - br}{k} \pmod{p-1}$ implies that

$$br \equiv h(m_1) - ks_1 \pmod{p-1} .$$

This gives $(p-1, r)$ solutions for b. Choose the one that gives $B \equiv \gamma^b \pmod{p}$. In this way we have found Bob's private key b as well as the exponent k that he is using for signatures.

21: BIT COMMITMENT

Alice and Bob agree to decide some matter by tossing a coin. Alice tosses the coin and Bob calls. Suppose, however, they they are in different rooms and so Bob can not see Alice tossing the coin. If they do not trust one another then they need a way to ensure that Bob does not change his call when he hears the result, nor Alice change the reported result when she knows Bob's guess.

One way to do this is for Bob to write down his guess and put it in a sealed envelope, which he gives to Alice. Then Alice tosses the coin and reports the result. Together they open the envelope and see if Bob's guess was correct. This is called a *commit and reveal* process. Bob commits his guess to paper and then, after Alice has announced the result, he reveals it to Alice.

Similar problems arise when sending messages. For example, if someone is selling racing tips but needs to ensure that he is paid before revealing the tip. Or if a poll is being conducted online where the results will be revealed to all participants but only after everyone has voted. We can use a similar commit and reveal process. It is usually called *bit commitment.* Bob chooses a bit 0 or 1 and commits this to a message sent to Alice. However the message is sent in such a way that Alice can not read it without further information and yet Bob can not alter his choice. There are a variety of methods for achieving this.

21.1 Bit Commitment using a Public Key Cipher

Let e_B and d_B be the encrypting and decrypting functions for a public key cipher used by Bob. The encrypting function e_B is published but the decrypting function d_B is kept secret by Bob. Now Bob makes his choice m from some large alphabet $\mathcal{A}$ to indicate his guess, and commits to Alice the encrypted message $c = e_B(m)$. Provided that the cipher is secure, Alice can not decipher this. When the time comes to reveal his choice, Bob sends to Alice his private key so that she can find the decrypting function d_B. Now she can compute

$$d_B(c) = d_B(e_B(m)) = m$$

and find out what Bob's choice was. She can also check that d_B and e_B are inverse functions so she can be confident that Bob has not sent the wrong private key so as to pretend his choice was different.

Note that Bob should not simply use $m = 1$ for a head and $m = 0$ for a tail. If he did this, Alice could simply calculate $e_B(0)$ and $e_B(1)$ and see which Bob had sent. This is a dictionary attack, where Alice can simply find all possible messages and compute the corresponding ciphertexts using the public key e_B. Instead, Bob should introduce some randomness. For example, he can pad the message 0 or 1 by following it with a large number of random bits. Then there are a very large number of possible messages and an equally large number of possible ciphertexts for Alice to decipher.

21.2 Bit Commitment using a Noisy Channel

We can also use our earlier work on noisy channels to give an alternative method for bit commitment. Suppose that Alice and Bob have two ways to communicate. They can use a clear channel or a noisy channel. The clear channel transmits without errors but the noisy channel is a binary symmetric channel with error probability $0 < p < \frac{1}{2}$. We assume that the noisy channel corrupts bits independently of any action by Alice or Bob, so neither can affect its behaviour.

Now Alice chooses a linear binary code with codebook $C \subset \mathbb{F}_2^N$ and minimum distance d. Bob chooses a random, non-trivial, linear map $\theta : C \to \mathbb{F}_2$. Both publish their choices. In order to send a bit $m \in \mathbb{F}_2$, Bob chooses a random code word $\boldsymbol{c} \in C$ with $\theta(\boldsymbol{c}) = m$ and sends this to Alice via the noisy channel. So Alice receives a vector $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ in which certain components of $\boldsymbol{c}$ have been altered.

The expected value for $d(\boldsymbol{e}, \mathbf{0})$ is Np and N is chosen so that this is very much larger than d. This means that Alice can not tell what the original codeword $\boldsymbol{c}$ was and hence can not find $\theta(\boldsymbol{c}) = m$ and determine Bob's choice.

When the time comes for Bob to reveal his choice, he sends the codeword $\boldsymbol{c}$ to Alice via the clear channel. Alice can now check that $d(\boldsymbol{c}, \boldsymbol{r})$ is close to the expected value Np. If it is, she accepts that $\theta(\boldsymbol{c})$ was Bob's choice. If not, then she rejects it. There is, of course, a small chance that many more or many fewer bits of $\boldsymbol{c}$ were corrupted by the noisy channel and so Alice rejects Bob's choice even though he has not cheated. We choose the parameters N, d so that the probability of this occurring is very small. If it does occur, Alice and Bob should just repeat the entire process.

We have seen that Alice can not read Bob's guess until he has revealed it. We also want to show that Bob can not cheat by changing his guess and sending a different code word $\boldsymbol{c}' \neq \boldsymbol{c}$ to Alice via the clear channel. Bob knows the code word $\boldsymbol{c}$ that he originally chose but he does not know how it was corrupted by the noisy channel. So, all he knows is that the vector $\boldsymbol{r}$ received by Alice is at a Hamming distance of about Np from $\boldsymbol{c}$. If he sends $\boldsymbol{c}'$, then he must ensure that $d(\boldsymbol{r}, \boldsymbol{c}') \approx Np$. The probability that this happens is small unless he chooses $\boldsymbol{c}'$ very close to $\boldsymbol{c}$. However, any two codewords are at least distance d apart, so we see that Bob can not cheat.

Now let us prove these results more carefully. First fix a small number q that is an acceptable probability of error. We will choose the linear code at random by specifying its syndrome matrix. Let S be an $(N-K) \times N$ matrix over $\mathbb{F}_2$ with each column chosen independently and uniformly from $\mathbb{F}_2^{N-K}$. There are $(2^{N-K})^N = 2^{(N-K)N}$ choices for S. Let C be the linear code book given by

$$C = \{\boldsymbol{x} \in \mathbb{F}_2^N : S\boldsymbol{x} = \mathbf{0}\} \ .$$

Consider a non-zero vector $\boldsymbol{c} \in \mathbb{F}_2^N$ with weight $d(\boldsymbol{c}, \mathbf{0}) = w$. Then $\boldsymbol{c}$ is in the code C when the w columns of S corresponding to the non-zero entries of $\boldsymbol{c}$ sum to $\mathbf{0}$. Hence there are $(2^{N-K})^{N-1} = 2^{(N-K)(N-1)}$ such matrices S. The number of vectors $\boldsymbol{c}$ with $d(\boldsymbol{c}, \mathbf{0}) < d$ is $V(N,d) = \sum_{j<d} \binom{N}{j}$. So the probability that our random code has minimum distance at least d exceeds

$$\frac{2^{(N-K)N} - V(N,d)2^{(N-K)(N-1)}}{2^{(N-K)N}} = 1 - \frac{V(N,d)}{2^{N-K}} \ .$$

Proposition 7.8 shows that we can find a linear code with minimum distance d provided that

$$h(d/N) < 1 - \frac{K}{N} \ .$$

This means that we can find codes with minimum distance at least δN for $0 < \delta < \frac{1}{2}$.

The code word $\boldsymbol{c}$ was sent by Bob but $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ was received. Here the entries (e_i) of $\boldsymbol{e}$ are independent random variables each taking the value 1 with probability p and 0 with probability $1-p$. So

$$d(\boldsymbol{c}, \mathbf{0}) = \sum_{i=1}^{N} e_i \sim \mathrm{Ber}(N, p)$$

is a Bernoulli random variable with mean Np and variance $Np(1-p)$. Chebyshev's inequality gives

$$\mathbb{P}\left(|d(\boldsymbol{c}, \mathbf{r}) - Np| \geqslant N\varepsilon\right) \leqslant \frac{p(1-p)}{N\varepsilon^2} \ .$$

We will choose N and ε so that this is less than q. Hence

$$\mathbb{P}\left(|d(\boldsymbol{c}, \boldsymbol{r}) - Np| \geqslant N\varepsilon\right) < q \ .$$

Alice will accept Bob's choice provided that $|d(\boldsymbol{c}, \boldsymbol{r}) - Np| < N\varepsilon$. So the probability of Alice incorrectly rejecting the choice is at most q.

Suppose that Bob tried to send another code word $\boldsymbol{c}'$ in place of $\boldsymbol{c}$ with $d(\boldsymbol{c},\boldsymbol{c}')=u$. Let

$$U=\{i : c_i \neq c_i'\} \qquad \text{and} \qquad V=\{i : c_i = c_i'\}$$

so that

$$|U| = d(\boldsymbol{c},\boldsymbol{c}') = u \qquad \text{and} \qquad |V| = N-u\ .$$

Then we have $\boldsymbol{r}-\boldsymbol{c}' = \boldsymbol{e} + (\boldsymbol{c}-\boldsymbol{c}')$ so

$$d(\boldsymbol{c}',\boldsymbol{r}) = \sum_{i\in U}(1-e_i) \ + \ \sum_{i\in V} e_i \quad \sim \quad \mathrm{Ber}(u,1-p)+\mathrm{Ber}(N-u,p)\ .$$

This is a random variable with mean $u(1-p)+(N-u)p = Np+u(1-2p)$ and variance $u(1-p)p+(N-u)p(1-p) = Np(1-p)$. Hence we have

$$\mathbb{P}\left(|d(\boldsymbol{c}',\boldsymbol{r})-Np-u(1-2p)| \geqslant N\varepsilon\right) < q\ . \tag{$*$}$$

Choose ε so that

$$\frac{2\varepsilon}{1-2p} < \delta$$

and then choose N so large that

$$\frac{p(1-p)}{N\varepsilon^2} < q\ .$$

Alice will only accept the code word $\boldsymbol{c}'$ if $|d(\boldsymbol{c}',\boldsymbol{r})-Np| < N\varepsilon$. Inequality $(*)$ shows that the probability this happens is less than q unless

$$u(1-2p) < 2N\varepsilon\ .$$

Hence we must have

$$d(\boldsymbol{c},\boldsymbol{c}') = u < \frac{2N\varepsilon}{1-2p} < \delta N\ .$$

Since the code words $\boldsymbol{c}$ and $\boldsymbol{c}'$ are different, we must have

$$d(\boldsymbol{c},\boldsymbol{c}') \geqslant \delta N$$

so we have a contradiction. This shows that Alice will correctly accept Bob's choice and Bob will be unable to cheat except with a very small probability q.

21.3 Semantic Security

We have shown how to produce ciphers that are secure in the sense that an enemy who knows the ciphertext can not find the corresponding plaintext in polynomial time. However, we have not considered whether the enemy can, nonetheless, obtain some partial information about the plaintext. When this too is impossible, then the cipher is *semantically secure*. This means, in particular, that it is impossible to decipher even a single bit of the ciphertext.

Semantically secure ciphers have been produced in recent years by encrypting the plaintext randomly. So, rather than replacing a single letter $a \in \mathcal{A}$ by a fixed codeword $c(a)$, we choose from a large set of possible codewords for a. This makes it very much harder to find a and hence to break the cipher.

Goldwasser and Micali gave an example of such a scheme that uses quadratic residues. Recall that a number k with $(k,N)=1$ is a *quadratic residue modulo* N if $k \equiv x^2 \pmod{N}$ for some x. It is a *quadratic non-residue modulo* N when there is no such x. For a prime p, we know that there is a primitive root γ modulo N, so k is a quadratic residue modulo p when $k=\gamma^{2r}$ for some integer r. This gives Euler's criterion: k is a quadratic residue modulo the prime p if and only if $k^{(p-1)/2} \equiv 1 \pmod{p}$. Now consider $N=pq$ the product of two distinct primes, p and q. The Chinese remainder theorem shows that k is a quadratic residue modulo N if and only if it is a quadratic residue modulo p and modulo q. This means that it is simple to determine whether k is a quadratic residue provided that we know how to factorize N.

Bob wishes to send to Alice a binary message $m = a_1a_2\dots a_K$ where each $a_j \in \mathbb{F}_2$. First Alice needs to set a key. She selects two different large primes p, q and sets $N = pq$. Then she chooses a random integer y modulo N with y a quadratic non-residue modulo N. It is easy to find such a y since Alice knows the factors of N. She publishes (N, y) as her public key.

Now Bob encrypts his message m as $e(a_1)e(a_2)\dots e(a_K)$ where

$$e(a_j) \equiv \begin{cases} x_j^2 \pmod{N} & \text{when } a_i = 0 \\ yx_j^2 \pmod{N} & \text{when } a_i = 1\ . \end{cases}$$

Here x_j is coprime to N and is chosen randomly for each letter a_j. Note that the length of the ciphertext is very much longer than that of the plaintext, since each bit of m is encrypted as a string of $\log_2 N$ bits.

To decrypt the ciphertext, we need to decide if each letter $c_j = e(a_j)$ is a quadratic residue modulo N, in which case $a_j = 0$, or a quadratic non-residue, in which case $a_j = 1$. This is simple for Alice since she knows the factors p and q of N. Using Euler's criterion she can determine whether c_j is a quadratic residue or non-residue modulo p and q. Then it is only a quadratic residue modulo N if it is a quadratic residue modulo both p and q.

If an enemy tries to decipher the ciphertext without knowing the factors p and q, then the problem becomes very hard. It is thought that this is as hard as finding the factors. Goldwasser and Micali proved that this scheme is semantically secure against an enemy who has polynomially bounded resources, at least provided that there is no polynomial time algorithm for determining whether a number is a quadratic residue modulo N.

Random encryption like the Goldwasser – Micali scheme gives greatly improved security but also greatly expands the length of the ciphertexts.