Chapter 1

Error correcting codes

1 Motivation

(1.1) Almost all systems of communication are confronted with the problem of noise. This problem has many different causes both human or natural and imply that parts of the messages you want to send, are falsely transmitted. The solution to this problem is to transmit the message in an encoded form such that it contains a certain amount of redundancy which enables us to correct the message if some errors have occurred.

(1.2) Mathematically one can see a coded message as a sequence of symbols chosen out of a finite set $\mathcal{F}$. If an error occurs during the transmission, some of the symbols will be changed, and the sequence received will differ at some positions from the original. If one would allow to transmit all possible sequences, one could never correct any errors because one could always assume that the received message is the one sent and that no errors have occurred.

(1.3) To avoid this problem we restrict ourself to transmit only special sequences which we call *code words*. If few errors occur during the transmission one will see that the received message is not a codeword. In this way the receiver knows that errors have occurred. To correct the message one searches for the code word which is the closest to the received message, i.e. which differs at the least number of positions with the received message.

(1.4) Example.
Suppose you have to navigate at distance a blind person through a maze. One could use a morse like coding system:

Action	Code
One step forward	..
One step backward	−−
Turn left	.−
Turn right	−.

This code is useful as long as no errors occur, because even if only a single error occurs the blind person would not notice this, understand a different command and probably walk against a wall.
To improve the system one could use this code

Action	Code
One step forward	...
One step backward	− − .
Turn left	. − −
Turn right	−.−

All the codewords here contain an even number of bars, if one error occurs during the transmission, there will be one bar more or less and the received message will no longer be a code word. The blind person will notice this and could ask for retransmission. However he will not be able to correct the message himself because if he receives −.., the original message could be ... or −.−. We call such a code a single error detecting code.
To do better we use the concatenation of the previous two codes.

Action	Code
One step forward	.....
One step backward	− − − − .
Turn left	. − . − −
Turn right	−. − .−

If here one error occurs, the blind person can deduce what the original message was because if the error occurred in the first two symbols he will see that the last three digits form a code word of the second code and decode the last three symbols. If the error occurs in the last tree digits these will no more be a code word of the second code, so he knows that the first two digits are correct and take these to decode. This code is a single error correcting code.
We 've seen that we can construct codes that are able to correct mistakes, but the price for it is redundancy. As we saw above we had to use five symbols to transmit the message, in stead of two, to produce a single error correcting code.

We will now formalize all this to get an exact definition of the concept code:

(1.5) Definition.
Suppose $\mathcal{F}$ is a finite set of symbols. A *n-code C* over $\mathcal{F}$ is a subset of $\mathcal{F}^n$. If $\mathcal{F} = \mathbb{F}_2 = \{\mathtt{0}, \mathtt{1}\}$ we call the code *binary*. If $|\mathcal{F}| = q$ we speak of a q-ary code.

So for the last example the used code

$$C := \{\mathtt{00000},\ \mathtt{11110},\ \mathtt{01011},\ \mathtt{10101}\}$$

is a binary 5-code.

(1.6) Definition.
An *error* occured at the i^{th} place changes the i^{th} entry of a codeword. e.g.

$$11110 \longmapsto 10110$$

is an error occurred at the second place.

(1.7) Definition.
The *Hamming distance* $d(\mathtt{x}, \mathtt{y})$ between two sequences of symbols $\mathtt{x}, \mathtt{y}$ is the number of places where they differ e.g.

$$d(\mathtt{11110}, \mathtt{01011}) = 3$$

This function is really a distance function meaning that it satisfies the following easily to prove properties

- $\forall \mathtt{x}, \mathtt{y} \in C : d(\mathtt{x}, \mathtt{y}) = d(\mathtt{y}, \mathtt{x})$
- $\forall \mathtt{x}, \mathtt{y} \in C : d(\mathtt{x}, \mathtt{y}) = 0 \iff \mathtt{x} = \mathtt{y}$
- $\forall \mathtt{x}, \mathtt{y}, \mathtt{z} \in C : d(\mathtt{x}, \mathtt{y}) \leq d(\mathtt{x}, \mathtt{z}) + d(\mathtt{z}, \mathtt{y})$

When i errors occur the Hamming distance between the original code word and the received message will be i. The normal procedure to correct the received message is to assume that the least possible errors occurred so the reconstructed codeword will be the one with the smallest Hamming distance from the received message.

(1.8) Example.
If we are still using code C and you receive as message `11010` you will

decode it as 11110 because

$$\begin{aligned} d(\texttt{11010}, \texttt{00000}) &= 3 \\ d(\texttt{11010}, \texttt{11110}) &= 1 \\ d(\texttt{11010}, \texttt{01011}) &= 3 \\ d(\texttt{11010}, \texttt{10101}) &= 4 \end{aligned}$$

and it is most likely that only 1 error had occurred. If the chance of transmitting a correct bit is 90% then the probability that only one error occurred is

$$\binom{5}{1}(.9)^4(.1)^1 = .32805$$

probability that 3 errors occurred:

$$\binom{5}{3}(.9)^2(.1)^3 = .0081$$

is 400 times less so it will be most obvious to decode it the way we did.

2 Definitions

(1.9) Definition.
The distance of a code is the minimum of the distances between all codewords.

$$d(C) = \{d(\texttt{x}, \texttt{y}) | \texttt{x}, \texttt{y} \in C\}$$

The parameters of a code are formed by n the number of symbols used for a code word, $|C|$ the size of the code an d its minimal distance. In this case we speak of an $(n, |C|, d)$-code

One can easily compute that in the example we are using $d(C) = 3$ and hence C is a $(5, 4, 3)$-code.

(1.10) Lemma.
A code C detects (up to) s errors if and only if $d(C) > s$.

Proof. Suppose that $d(C) > s$. Let $\texttt{c}$ be a codeword in C, and let $\texttt{x}$ be a word obtained from $\texttt{c}$ by making up to s errors. Now $d(\texttt{x}, \texttt{c}) \leq s$, and so if $s < d(C)$ then $\texttt{x}$ cannot be a codeword. So C detects up to s errors.
Conversely if $s \geq d(C)$ then there exist codewords $\texttt{c}_1$ and $\texttt{c}_2$ in C with $(\texttt{c}_1, \texttt{c}_2) = d(C) \leq s$. So $\texttt{c}_1$ can be converted into $\texttt{c}_2$ with no more than s errors, and this will not be detected. $\square$

(1.11) Lemma.
A code C corrects (up to) t errors if and only if $d(C) \geq 2t+1$.

Proof. Suppose that $d(C) > 2t$. Let $\mathtt{c}$ be a codeword in C, and suppose that a word $\mathtt{x}$ has been obtained from $\mathtt{c}$ with at most t errors. We need to show that the received word $\mathtt{x}$ is closer to our codeword $\mathtt{c}$ than to any other codeword $\mathtt{r}$. Now by the Triangle Inequality, we have

$$\begin{aligned} d(\mathtt{c},\mathtt{x}) &\leq d(\mathtt{c},\mathtt{r}) + d(\mathtt{r},\mathtt{x}) \\ -d(\mathtt{c},\mathtt{r}) + d(\mathtt{c},\mathtt{x}) &\leq +d(\mathtt{r},\mathtt{x}) \\ 2t+1-t &\leq d(\mathtt{x},\mathtt{r}) \\ t+1 &\leq d(\mathtt{x},\mathtt{r}) \end{aligned}$$

The converse is left as an exercise. □

(1.12) That the concept of error correcting codes is indeed very useful to transmit messages over a noisy channel can be shown by a little computation. Suppose again that the error rate is 10% and that we have to transmit messages, each chosen out of 4 possibilities. If we use the simplest code

$$C = \{\mathtt{00}, \mathtt{11}, \mathtt{01}, \mathtt{10}\}$$

the probability that one receives the correct message is here.

$$.9^2 = 81\%$$

If we use the code

$$C := \{\mathtt{00000},\ \mathtt{11110},\ \mathtt{01011},\ \mathtt{10101}\}$$

the probability of finding the correct message will be

$$.9^5 + \binom{5}{1}.9^4.1 = 91\%.$$

So the error rate is reduced by 50%.

(1.13) Aside.
Richard W. Hamming Richard W. Hamming received the 1968 Turing Award for his work on error-correcting codes at AT& T Bell Laboratories, where he worked for 30 years on a multitude of research projects. He joined the Naval Postgraduate School in Monterey, California, in 1976 as Adjunct Professor, where he is currently involved in teaching and writing books on probability and combinatorics. Hamming is a scientific generalist whose aims have included to teach his students an attitude of excellence toward science, not just technical skills.

Richard W. Hamming

3 Examples

(1.14) Example. Repetition codes
Look at the binary code

$$\{\texttt{000}, \texttt{111}\}$$

This is called a $[3, 1]$-repetition code. In this notation, the first number n is the length of each codeword, and the second number k is the length of the unit being repeated. (So k must divide n for the notation to make sense!) The number of times each unit is repeated is $r = n/k$.
Another example is the $[4, 2]$-repetition code. The original message units have length 2; they are 00, 01, 10 and 11. The 4-digit codewords are formed by simply repeating them, so we get 0000, 0101, 1010 and 1111 respectively.

Suppose we have a codeword for a $[n, k]$-repetition code. It will look like

$$\underbrace{d_1 d_2 \cdots d_k \; d_1 d_2 \cdots d_k \;\; \cdots \;\; d_1 d_2 \cdots d_k}_{r},$$

where the d_i are digits in $\mathcal{F}$.
The number of codewords is q^k where q is the size of the alphabet, and the length of a codeword is n. The minimal distance of the code is n/k because if two codewords differ at place i then they also differ at the n/k places $i + zk$ with $-i/k \le z \le (n - i)/k$.

(1.15) Example. Belgium bank accounts
A Belgian Bank Account number consists of 12 digits. Grouped in a group of 3 one of 7 and one of 2. E.g.

$$\texttt{103} - \texttt{0202707} - \texttt{45}$$

Only those numbers are allowed such that rest of the number formed by the first 10 after division by 97 equals the last two digits:

$$\texttt{1030202707} = \texttt{10620646} \cdot \texttt{97} + \texttt{45}.$$

Show that this code is one error detecting but not one error correcting. What are the parameters of this code?

(1.16) Example. EAN-13
EAN-13 is used world-wide for marking retail goods. The symbol encodes 13 characters: the first two or three are a country code which identify the country in which the manufacturer is registered (not necessarily where the

product is actually made). The country code is followed by 9 or 10 data digits (depending on the length of the country code) and a single digit checksum.
The checksum is a modulo 10 calculation:

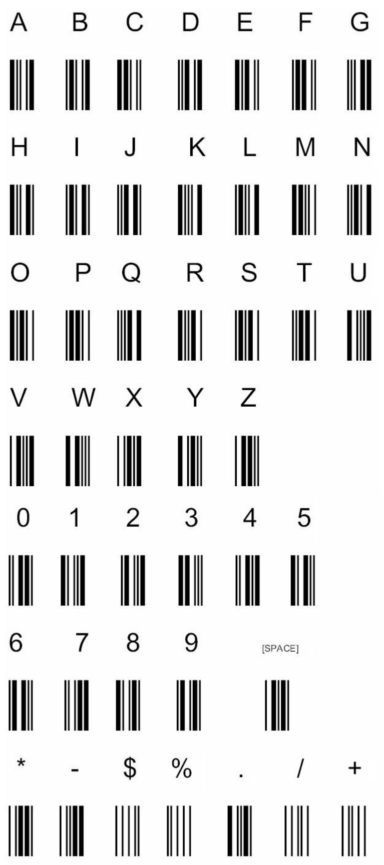
1. Add the values of the digits in the even-numbered positions.
2. Multiply this result by 3.
3. Add the values of the digits in the odd-numbered positions.
4. Sum the results of steps 2 and 3.
5. The check character is the smallest number which, when added to the result in step 4, produces a multiple of 10.

Example: Assume the barcode data = `001234567890`

1. $0 + 2 + 4 + 6 + 8 + 0 = 20$
2. $20 \cdot 3 = 60$
3. $0 + 1 + 3 + 5 + 7 + 9 = 25$
4. $60 + 25 = 85$
5. $85 + X = 90$, therefore $X = 5$ and the codeword is `0012345678905`.

Show that this code is one error detecting but not one error correcting. What are the parameters of this code?

(1.17) Example. Code 39
This code has an alphabet of two letter $\{\mathtt{W}, \mathtt{n}\}$ standing for wide and narrow. The code has 45 codewords that correspod to the digits 0-9, the letters A-Z (upper case only), and the following symbols: space, minus (-), plus (+), period (.), dollar sign ($), slash (/), percent (%) and a special start/stop character.
Each codeword consists of 9 elements and includes exactly 3 wides. E.g. The code representing the letter A is: `WnnnnWnnW`. Usually the codeword is printed by alternating black and white bars (which according to the code can be wide and narrow).



What are the parameters of this code?

(1.18) Example. A parity code
We can represent the alphabet using binary strings of length 5 as follows. If we wish, we can use the 6 remaining 5-digit strings for punctuation marks or other characters.

Character	Code	Character	Code
A	00000	N	01101
B	00001	O	01110
C	00010	P	01111
D	00011	Q	10000
E	00100	R	10001
F	00101	S	10010
G	00110	T	10011
H	00111	U	10100
I	01000	V	10101
J	01001	W	10110
K	01010	X	10111
L	01011	Y	11000
M	01100	Z	11001

Suppose that we encode our 5-digit strings by adding a 6th digit, which we choose such that the total number of 1s in the string is even. So J $=$ `01001` becomes `010010`, and V $=$ `10101` becomes `101011`.
We now have a code, where the codewords are the 6-digit strings with even *parity*. That is, the sum of their digits is an even number.

Suppose a single error occurs in the transmission of a codeword. Then either a `1` is changed to `0`, or a `0` is changed to `1`. In either case sum of the digits is changed by 1. So the parity changes.
This means that any single error will be detected. In fact, any odd number of errors will be detected, but any even number of errors will be missed (since if the parity changes an even number of times, it is the same as it was originally.)
On the other hand, this code has no capability for error-correction. If a digit is transmitted incorrectly, we have no way of knowing which digit it was, since changing any of the digits will restore the original parity.

(1.19) Example. Hamming's square code
This code extends the idea of parity checking to allow single errors to be corrected. We start with message units which are binary strings of length 4. To encode a message unit, we write the digits in a square array, and compute the sums of rows and columns $\bmod 2$.

$$\texttt{1101} \longmapsto \begin{array}{cc|} 1 & 1 \\ 0 & 1 \\ \hline \end{array} \longmapsto \begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & 1 \\ \hline 1 & 0 & 1 \end{array}$$

The codeword is formed by combining the three rows of this array. So for the example above our codeword is `110011101`. To decode we write the received message as a 3×3 array and look at the upper left 2×2 block; if no errors have occurred, this will contain the original 4-bit message unit.

How can this code correct an error? Suppose you receive the message `100011101`. Put it into a square array as follows.

$$\begin{array}{cc|c} \mathtt{1} & \mathtt{0} & \mathtt{0} \\ \mathtt{0} & \mathtt{1} & \mathtt{1} \\ \hline \mathtt{1} & \mathtt{0} & \mathtt{1} \end{array}$$

The parity check in the first row is incorrect, so we know that there is an error in this row. The check in the second column is also incorrect and so there is an error in this column. So assuming that only one error has occurred, we know that it must be the digit in the first row and second column which is wrong.

How many errors will this code detect? Suppose that errors have turned a codeword into a different codeword. If a particular digit has been affected, then since the row and column sums tally, there must be another error in the same row, and one in the same column. But these give, respectively, another column and another row in which the checks would be wrong, unless there was a fourth error at the "opposite corner" of the square formed by the three errors we know about. So there must be at least four errors in the code. It follows that three or fewer errors will be detected by the code.

3.1 Equivalence of codes

Consider the following three codes, all subsets of ${\mathbb{F}_2}^6$.

$$\begin{aligned} C_1 &= \{\mathtt{001100}, \mathtt{011011}, \mathtt{110011}\} \\ C_2 &= \{\mathtt{000110}, \mathtt{101101}, \mathtt{111001}\} \\ C_3 &= \{\mathtt{100001}, \mathtt{110110}, \mathtt{011110}\} \end{aligned}$$

C_2 and C_3 are obtained from C_1 by permuting the order of the digits; in C_2 they have been cyclically shifted one position to the right, and in C_3 the first and third symbols and the fourth and sixth symbols have been switched. We say that C_2 and C_3 were obtained from C_1 by means of a *positional permutation.*
It is clear that the Hamming distance between codewords will be the same, whether we work in C_1, C_2 or C_3. In general, we have the following lemma.

(1.20) Lemma.
Performing a positional permutation on the words of a code preserves the distance between codewords, and so preserves the minimum distance of the code.

Now consider the following codes in ${\mathbb{F}_2}^6$.

$$\begin{aligned} C_1 &= \{\texttt{011011}, \texttt{010101}, \texttt{000111}\} \\ C_2 &= \{\texttt{010011}, \texttt{011101}, \texttt{001111}\} \\ C_3 &= \{\texttt{011100}, \texttt{010010}, \texttt{000000}\} \end{aligned}$$

This time the changes have been made by permuting the symbols of the alphabet at particular positions in the words. C_2 is obtained from C_1 by changing the digit in the third position of each codeword. C_3 is obtained by changing the last three digits. We say that C_2 and C_3 were obtained by C_1 by means of *symbolic permutations*.
Here is another example of symbolic permutation. Here the codes are all subsets of ${\mathbb{F}_3}^6$.

$$\begin{aligned} C_1 &= \{\texttt{012112}, \texttt{011022}, \texttt{120021}\} \\ C_2 &= \{\texttt{112111}, \texttt{111021}, \texttt{220020}\} \\ C_3 &= \{\texttt{102002}, \texttt{100122}, \texttt{021120}\} \end{aligned}$$

C_2 is obtained from C_1 by applying the permutation $0 \longmapsto 1 \longmapsto 2 \longmapsto 0$ in the first position, and the inverse of that permutation in the last position. C_3 is obtained from C_1 by performing the same symbolic permutation on every position: the permutation which switches 0 and 1, leaving 2 unchanged.
These symbolic permutations are certainly distance preserving, so we have the following lemma.

(1.21) Lemma.
Performing a symbolic permutation, at some or all of the positions of a code, preserves the distance between codewords and hence the minimum distance of the code.

(1.22) Definition.
Two codes are *equivalent* if one can be obtained from the other by means of a sequence of positional and symbolic permutations.

Clearly all equivalent codes have the same minimum distance. But the converse is false; it is not true that all codes of word length n, size M and with the same minimum distance d are equivalent.

(1.23) Example.

$$\begin{aligned} C_1 &= \{0000, 0011, 0101, 0110\} \\ C_2 &= \{0111, 1011, 1101, 1110\} \end{aligned}$$

It is easily checked that these two binary codes are both $(4, 4, 2)$ codes. In fact, they both have the property that the Hamming distance between any pair of codewords is 2. But these two codes are not equivalent. (Exercise: why not?)

(1.24) Example.
Consider the Hamming distance between pairs of codewords for each of the following codes, X and Y.

$$\begin{aligned} X &= \{x_1,\ x_2,\ x_3,\ x_4\} \\ &= \{0000000, 0011001, 0101011, 0110111\} \\ Y &= \{y_1,\ y_2,\ y_3,\ y_4\} \\ &= \{1011001, 1110111, 1101011, 0111000\} \end{aligned}$$

X	x_1	x_2	x_3	x_4
x_1		3	4	5
x_2			3	4
x_3				3

Y	y_1	y_2	y_3	y_4
y_1		4	3	3
y_2			3	5
y_3				4

Note that these tables contain the same numbers but in a different order. If we rearrange the order of words in Y then we obtain an identical table to that for X.

Y	y_4	y_1	y_3	y_2
y_4		3	4	5
y_1			3	4
y_3				3

We say that X and Y above are *distance isomorphic*. Here is a formal definition.

(1.25) Definition.
Two p-ary codes C_1 and C_2 of the same length and size are said to be distance isomorphic if their words can be ordered in such a way that, for all $x_i, x_j \in C_1$ and all $y_i, y_j \in C_2$, we have $d(x_i, x_j) = d(y_i, y_j)$.

Equivalent codes are necessarily distance isomorphic, but not vice versa.

Chapter 2

Linear codes

In order to use more advanced algebraical methods, we will now define the notion of a linear code.

1 Definition and examples

(2.1) Definition.
A *linear* $[n, m, d]$*-code* C over the finite field $\mathbb{F}_q$ is an m-dimensional subspace of $\mathbb{F}_q^n$, which has a minimal distance d. By subspace we mean that

$$\forall \mathrm{x}, \mathrm{y} \in C : \forall a, b \in \mathbb{F}_q : a\mathrm{x} + b\mathrm{y} \in C$$

The fact of being m-dimensional makes that C contains q^m code words.

NB. For linear codes we use square brackets to describe the parameters. The only difference between the square brackets and the round is that the middle parameter of the former is the dimension of the code as a vector space while the middle parameter of the latter is the number of codewords. This implies that a linear $[n, m, d]$ code over $\mathbb{F}_q$ is also an ordinary (n, q^m, d) code over $\mathbb{F}_q$.

(2.2) Example.

- $C := \{\mathtt{000}, \mathtt{110}, \mathtt{011}, \mathtt{101}\}$ is a binary linear $[3, 2, 1]$-code.
- $C := \{\mathtt{00000},\ \mathtt{11110},\ \mathtt{01011},\ \mathtt{10101}\}$ is a binary linear $[5, 2, 3]$-code.
- $C := \{\mathtt{0000},\ \mathtt{1110},\ \mathtt{0101},\ \mathtt{1001}\}$ is not linear because $\mathtt{1110} + \mathtt{0101} = \mathtt{1011} \notin C$.
- $C := \{\mathtt{0000},\ \mathtt{1021},\ \mathtt{2012},\ \mathtt{0111},\ \mathtt{0222},\ \mathtt{1102},\ \mathtt{2201},\ \mathtt{1210},\ \mathtt{2120}\}$ is a linear $[4, 2, 3]$-code over $\mathbb{F}_3$.
- $C := \{\mathtt{00}, \mathtt{1}\xi, \xi\xi^2, \xi^2\mathtt{1}\}$ is a linear $[2, 1, 2]$-code over $\mathbb{F}_4$.

(2.3) Definition.
The weight w of a codeword is the total number of non-zero elements in its sequence:

$$w(\texttt{01101101}) = 5$$

(2.4) In a linear code C the minimal distance is also the least weight of the codewords in C because if $d = d(\mathtt{x}, \mathtt{y})$ then $\mathtt{x} - \mathtt{y}$ will be nonzero at exactly those places where $\mathtt{x}$ and $\mathtt{y}$ differ.

$$\forall \mathtt{x}, \mathtt{y} \in C : d(\mathtt{x}, \mathtt{y}) = w(\mathtt{x} - \mathtt{y}).$$

Because $\mathtt{x} - \mathtt{y}$ is in C there will be a vector which has weight equal to the minimal distance. Vice versa we have that $d(\mathtt{o}, \mathtt{x}) = w(x)$ so the minimal weight will be the minimal distance.

2 Basis and generator matrix

We will here review the concept of linearly dependence and see how it relates to linear codes. If we consider some codewords $\mathtt{x_1}, \cdots, \mathtt{x_k} \in C$ then they are said to be *linearly independent* if there exists no numbers $a_i \in \mathbb{F}_2$ such that

$$a_1\mathtt{x_1} + \cdots + a_k\mathtt{x_k} = 0$$

and not all the a_i are zero. Otherwise $\{\mathtt{x_i}\}$ is said to be *linear dependent*.

(2.5) Example.

$$\texttt{0101}, \texttt{0110}, \texttt{0011}$$

are linear independent because

$$a_1\texttt{0101} + a_2\texttt{0110} + a_3\texttt{0101} = (0, a_1 + a_2, a_2, a_3) = \texttt{0000}$$

implies that $a_1 = a_2 = a_3 = 0$. On the other hand is

$$\texttt{0111}, \texttt{1111}, \texttt{1110}, \texttt{1001}$$

linear dependent because

$$\texttt{0111} + \texttt{1110} + \texttt{1001} = \texttt{0000}$$

A subset $S \subset C$ is said to be *generating* if all codewords in C can be expressed as a linear combination of codewords in S. S is said to be a *base* if it is both linear independent and generating. If this is the case any codeword in C can be written as *unique* combination of base elements.

(2.6) Example.
If we consider the binary code

$$C := \{\texttt{00000},\ \texttt{11110},\ \texttt{01011},\ \texttt{10101}\}$$

then there are three possible bases containing all 2 elements

$$\begin{aligned} B_1 &:= \{\texttt{11110}, \texttt{01011}\} \\ B_2 &:= \{\texttt{10101}, \texttt{01011}\} \\ B_3 &:= \{\texttt{11110}, \texttt{10101}\} \end{aligned}$$

A very easy way to handle vector spaces is using matrices. Matrices over finite fields multiply just the way they do over the real numbers, except that you must use the calculation rules of the field (e.g. in $\mathbb{F}_2$ $1 + 1 = 0$).

(2.7) Example.

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1\cdot 1 + 0\cdot 1 + 1\cdot 1 & 1\cdot 0 + 0\cdot 1 + 1\cdot 1 \\ 1\cdot 1 + 1\cdot 1 + 0\cdot 0 & 1\cdot 0 + 1\cdot 1 + 0\cdot 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

Calculating the determinant of a matrix is also done in the ordinary way (remember that in $\mathbb{F}_2$, we can replace all the minus signs by plus signs).

$$\text{Det} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = 1\cdot 1 + 1\cdot 0 = 1$$

(2.8) Definition.
When we have an m-dimensional code C with a base in it then we can make a $m \times n$-matrix. The rows of this matrix consist of the base-vectors of the considered base. This matrix is called a *generator matrix*.

(2.9) We know that there are q^m code words so it is reasonable to consider the set of messages equal to the space $\mathbb{F}_q^m$. Encoding the message means that we assign to each message word a unique codeword. This can be done easily by considering the message as a row-matrix and multiplying it with the generator matrix. The encoding depends highly on the generator matrix, different generator matrices give different encodings but use the same codewords.

(2.10) In fact if we multiply the generator matrix on the left by an invertible $m \times m$-matrix we will obtain a new generator matrix because multiplying on the left corresponds to a base change in the C.

(2.11) One could also try to multiply on the right with an invertible matrix but doing this will alter the corresponding code, such that the decoding capacities will alter as well. However if we multiply on the right by a permutation matrix (i.e. a matrix which has in every row and every column exactly one 1 and all the other entries zero) this will correspond to a permutation of the columns of the generator matrix. This means that we just constructed a new code by switching some of the digits around i.e. we performed a positional permutation. This operation will not alter the minimal distance of the code as we have seen in chapter 1.
In chapter 1 we have also performed symbolic permutations on codes, however unlike positional permuations they can destroy the linear structure of the code. E.g. look at the binary code $\{\texttt{000}, \texttt{110}, \texttt{011}, \texttt{101}\}$. This is a linear code but if we switch 0 and 1 in all positions the new code $\{\texttt{111}, \texttt{001}, \texttt{100}, \texttt{010}\}$ is not linear anymore.
Therefore we call two linear codes *linearly equivalent* (or shortly equivalent) if they can be transformed into each other using only positional permutations.

(2.12) Example.
If we consider the code

$$C := \{\texttt{00000},\ \texttt{11110},\ \texttt{01011},\ \texttt{10101}\}$$

then there are 6 possible generator matrices two for each base

$$B_1 \longmapsto \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$B_2 \longmapsto \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$B_3 \longmapsto \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

And if we take an invertible 2×2-matrix and multiply it with one of these generator matrices

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

we obtain again one of the six generator-matrices. However if we multiply on the right with a permutation matrix

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

We obtain a generator matrix of a different code

$$C' := \{\texttt{00000},\ \texttt{11011},\ \texttt{00111},\ \texttt{11100}\}$$

This code is in fact equivalent to C and has thus the same minimal distance: 3.

(2.13) Definition.
A generator matrix G whose left part is the identity matrix, i.e. $G = [\mathbb{I}_m X]$, is said to be in *normal position*. For the code we considered in the previous example, is

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

a generator matrix in normal position. A code that admits a such generator matrix is called a *systematic code.*

A systematic code is easier to decode than an ordinary code, especially when no errors have occured. This is because if $\texttt{m}$ is the message we want to encode the corresponding codeword is

$$\texttt{m}G = \texttt{m}[\mathbb{I}X] = [\texttt{m}\texttt{m}X]$$

so the first m digits of the codeword are equal to the message itself.
Luckily we have the following theorem.

(2.14) Theorem.
Every linear code is linearly equivalent to a systematic code.

Proof. Take any generator matrix A for the code C. Perform the following algorithm

(2.15) Algorithm. ***Gauss-Jordan elimination***
Input: *An $m \times n$ matrix A.*
For $i = 1$ *to m do:*

swap *rows as necessary to ensure that the leftmost nonzero element starting from row i is in row i,*

divide *row i by this element.*

make zeroes in this column. *That is, where the first nonzero in row i is in column j, subtract an appropriate multiple of row i from each row below and above it so that there are zeros in column j below and above row i.*

After these steps we obtain an $m \times n$ matrix A' in **reduced row echelon form**. *From linear algebra we know that $A' = BA$ for some invertible matrix B because we only performed invertible row operations. In other words A' is also a generator matrix of C.*
construct the systematic code: *Let p_i denote the position of the first nonzero element of A' on row i. By our construction the column p_i is the zero column with only a one on row i. Now permute the colums in such a way that column p_i becomes column i. After this permutation the new matrix A'' is a generator matrix in normal position.*

$\square$

3 The parity check matrix

On the vector space C we can also define a scalar product $\cdot$. This maps every pair of codewords onto $\mathbb{F}_q$ in a bilinear way

(2.16) Definition.

$$\forall \mathtt{x} := \mathtt{x_1}\cdots\mathtt{x_n}, \mathtt{y} := \mathtt{y_1}\cdots\mathtt{y_n} \in \mathbb{F}_q^n : \mathtt{x}\cdot\mathtt{y} = \mathtt{x_1y_1} + \cdots + \mathtt{x_ny_n} \in \mathbb{F}_q$$

two sequences whose scalar product is 0 are said to be *orthogonal*. Note that we also have the following identity:

$$\forall \mathtt{x}, \mathtt{y}, \mathtt{z} \in \mathbb{F}_q^n : (\mathtt{x}+\mathtt{y})\cdot\mathtt{z} = \mathtt{x}\cdot\mathtt{z} + \mathtt{y}\cdot\mathtt{z}$$

The *orthogonal complement* of a subset $S \subset \mathbb{F}_q^n$ is the set of all sequences that are orthogonal to each sequence in S:

$$S^\perp := \{\mathtt{x} \in \mathbb{F}_q^n | \forall \mathtt{y} \in S : \mathtt{x}\cdot\mathtt{y} = 0\}$$

(2.17) Example.
Here we first compute the scalar product of two sequences in $\mathbb{F}_2$.

$$\begin{aligned}\mathtt{11001}\cdot\mathtt{01101} &= 1\cdot 0 + 1\cdot 1 + 0\cdot 1 + 0\cdot 0 + 1\cdot 1\\ &= 0+1+0+0+1 = 0\end{aligned}$$

We can also compute the orthogonal complement of the code

$$\begin{aligned}C &:= \{\mathtt{00000},\ \mathtt{11110},\ \mathtt{01011},\ \mathtt{10101}\}\\ C^\perp &:= \{\mathtt{00000},\ \mathtt{11110},\ \mathtt{01010},\ \mathtt{10100},\\ &\qquad \mathtt{11001},\ \mathtt{00111},\ \mathtt{10011},\ \mathtt{11001},\ \mathtt{01101}\}\end{aligned}$$

which is a again a linear code because of the distributivity of the scalar product:

$$\mathtt{x} \cdot \mathtt{y} = \mathtt{x} \cdot \mathtt{z} \implies \mathtt{x} \cdot (\lambda \mathtt{y} + \mu \mathtt{z}) = 0$$

This code is called the *orthogonal code* or *dual code*. In general if C is an $[n, m, d]$-code then its orthogonal code is an $[n, n-m, e]$-code, where e has to be determined by checking.

When we have a code C then $C^\perp$ is again a code, so we can consider a generator matrix of this code. If we call G the generator matrix of C and H a generator matrix of $C^\perp$ then the following identity holds:

$$GH^t = HG^t = 0$$

where $\cdot^t$ stands for the transposed of a matrix. We will call H the parity check matrix of the code C. If G is a generator matrix in standard form $[\mathbb{I}_m X]$ we can also choose a special form of the parity check matrix

$$H^t = \begin{pmatrix} -X \\ \mathbb{I}_{n-m} \end{pmatrix} \text{ because } \begin{pmatrix} 1_m & X \end{pmatrix} \begin{pmatrix} -X \\ 1_{n-m} \end{pmatrix} = (X - X) = 0$$

The parity check matrix is a very useful tool in decoding the code. Because H is a generator matrix of the orthogonal code, the following holds

$$\mathtt{c} \in C \iff H\mathtt{c}^t = \mathtt{o}$$

(2.18) Definition.
For a random word $\mathtt{x} \in \mathbb{F}_q^n$ we define its *syndrome* as

$$\mathtt{s}(\mathtt{x}) := H\mathtt{x}^t \in \mathbb{F}_q^{n-m}.$$

The codewords are the words with zero syndrome.

Suppose we receive a word $\mathtt{r}$ then we can assume that it is of the form $\mathtt{c} + \mathtt{e}$ where $\mathtt{c}$ is the transmitted codeword, and $\mathtt{e}$ is the error vector. So to remove the errors from $\mathtt{r}$ we have to find $\mathtt{e}$. If we look at the syndrome of $\mathtt{r}$ we see that it only depends on the error vector and not on the codeword. So instead of looking at the received word we should concentrate on the syndromes.
One should make a list of all possible syndromes and associate to each one the error vector with the least weight producing that syndrome, and thus the most likely to have occured. It is not always true that there is such a unique error vector, more than one error vector can have the same weight and syndrome. If this is the case we denote this also in the list.

(2.19) Algorithm. Encoding and decoding a linear code
Suppose that C is a systematic $[n, m, d]$-code and G is a generator matrix in normal position with H the corresponding parity check matrix.
Encoding
Take the message word $\mathtt{m} \in \mathbb{F}_2^m$ and multiply it with G and transmit it.
Decoding

1. *Make a list of syndromes and corresponding error vectors*
2. *Compute the syndrome of the received codeword $\mathtt{s} := H\mathtt{r}^t$.*
3. *Look in the list of syndromes which error vector $\mathtt{e}$ corresponds to $\mathtt{s}$. If there is only one code, subtract it from $\mathtt{r}$. The message word $\mathtt{m}$ is most likely to be the first m bits of $\mathtt{r} + \mathtt{e}$. If there are more error vectors, all those give equally probable message words, so if possible one should better ask for retransmission.*

(2.20) Example.
Consider the $[5, 2, 3]$-code with generator matrix

$$G := \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

The possible messages and their codewords are then

message	code word
00	00000
10	10110
01	01011
11	11101

The parity check matrix is

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

The list of possible syndromes and error vectors:

Syndrome	error vectors
000	00000
100	00100
010	00010
110	10000
001	00001
101	11000, 00101
011	01000
111	10001, 01100

Suppose we receive `11011`, computing the syndrome gives `s` := `110` so the error vector is `10000` and the corrected code word is `01011` and the message was `01`.
Suppose we receive `00111`, computing the syndrome gives `s` := `111`. There are two equally probable error vectors, so the code word could either be `10110` or `01011`.

Chapter 3

Perfect codes

1 Introduction

(3.1) One of the main challenges of coding theory is to find the best possible codes, but what are in fact the criteria for a good code? One wants to transmit as fast as possible, as many as possible information over a channel, such that there occur as few as possible mistakes.

(3.2) So for a linear $[n, m, d]$-code over a field $\mathbb{F}_q$ one wants to increase both the ratio m/n and d/n. As is expected one can not improve them both as much as one wants, because there are inequalities that are satisfied between those parameters.

(3.3) One of those inequalities is called the *sphere packing boundary*. If the minimum distance between two code words is d then one can draw around each codeword $\mathtt{c}$ in $\mathbb{F}_q^n$ a sphere with radius $(d-1)/2$.

$$B_{\mathtt{c}} := \{\mathtt{x} \in \mathbb{F}_q^n | d(\mathtt{x}, \mathtt{c}) \leq \frac{d-1}{2}\}$$

All those spheres are non-intersecting because if

$$\mathtt{x} \in B_{\mathtt{c}_1} \cap B_{\mathtt{c}_2}$$

then by the triangle inequality

$$d(\mathtt{c}_1, \mathtt{c}_2) \leq d(\mathtt{c}_1, \mathtt{x}) + d(\mathtt{x}, \mathtt{c}_2) \leq \frac{d-1}{2} + \frac{d-1}{2} \leq d-1$$

There are two code words at a distance less than d which is impossible.

In each of these spheres there are exactly

$$\sum_{0\leq i\leq \frac{d-1}{2}} \binom{n}{i}(q-1)^i$$

words. This sum corresponds to the number of words with $0, 1, \ldots, (d-1)/2$ errors: $\binom{n}{i} = \frac{n!}{(n-i)!i!}$ counts the possible ways to choose the i-error locations and $(q-1)^i$ counts the possible errors we can use in each location. For each sphere we have different elements and there are q^m spheres so the union of all balls contains

$$q^m\left(\sum_{0\leq i\leq \frac{d-1}{2}} \binom{n}{i}(q-1)^i)\right)$$

elements. There are only q^n elements so the expression above is smaller than q^n. From this we can conclude:

(3.4) Theorem. Sphere packing boundary
For every linear $[n, m, d]$-code over $\mathbb{F}_q$ we have that

$$\sum_{0\leq i\leq \frac{d-1}{2}} \binom{n}{i}(q-1)^i \leq q^{n-m}.$$

This warrants the following definition.

(3.5) Definition.
A linear $[n, m, d]$-code over $\mathbb{F}_q$ is perfect if the the balls B_c cover whole $\mathbb{F}_q^n$. This holds if and only if

$$\sum_{0\leq i\leq \frac{d-1}{2}} \binom{n}{i}(q-1)^i = q^{n-m}.$$

(3.6) In the previous chapter we have seen how to decode a linear code by using a table of syndromes. To every syndrome corresponded one or more error vectors. When we had a syndrome with more than one error vector, we could not uniquely decode the received message. In the case of perfect codes this last situation does not occur.

(3.7) Lemma.
A linear code is perfect if and only if for every syndrome there is a unique error vector of weight $\leq \frac{d-1}{2}$.

Proof. For every linear $[n, m, d]$-code there are always q^{n-m} syndromes. Error vectors inside the ball B_0 all have a different syndrome because the code is $\frac{d-1}{2}$ error correcting. By the pigeon-hole principle every syndrome has a unique error vector as soon as the number of elements in the ball equals the number of syndromes i.e. is the sphere packing boundary is reached. □

As we've seen perfect codes are in many ways very good, but unfortunately these codes are very rare. In what follows we will describe all perfect linear codes.
First notice that a code can never be perfect if its minimal distance is even. In this case, there would exist words that are at distance $d/2$ of two code words. so the syndrome decoding can't be unique.

2 Repetition codes

A trivial class of linear codes are the binary repetition codes.

(3.8) Definition.
A binary repetition code is a code consisting of two code words

$$C := \{\texttt{0} \ldots \texttt{0}, \texttt{1} \cdots \texttt{1}\}.$$

A repetition code has as parameters $[n, 1, n]$ and so it is easy to check whether such a code is perfect.

(3.9) Theorem.
A binary repetition code is perfect if and only if n is odd.

Proof. We already know that n has to be odd so $n = 2t + 1$. We can use the binomial of newton

$$\begin{aligned}\sum_{i=0}^{t} \binom{i}{n} (2-1)^i &= \sum_{i=0}^{t} \frac{1}{2} (\binom{n}{i} + \binom{n}{n-i})(1)^i \\ &= \frac{1}{2} \sum_{i=0}^{n} \binom{n}{i} = 2^{n-1}.\end{aligned}$$

□

3 Hamming codes

In this section, we consider an important family of perfect codes which are easy to encode and decode.
We first start with binary Hamming codes and afterwards we define them for arbitrary finite fields.

(3.10) Definition.
A binary code is a *Hamming code* if it has a parity check matrix $H \in \mathrm{Mat}_{r\times 2^r-1}(\mathbb{F}_2)$ consisting of all possible column vectors of $\mathbb{F}_2^r$.

$$\begin{pmatrix} v_1 & \dots & v_n \end{pmatrix},\ v_i \in \mathbb{F}_2^r - \{0\}$$

We denote this Hamming code as $\mathtt{Ham}(r,2)$.

(3.11) Example.
If r= 2 this means that

$$H := \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \implies G := \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

So $\mathtt{Ham}(2,2)$ is the binary repetition [3,1,3]-code. Notice that this is the only Hamming code that is a repetition code because $d = 3$ for a Hamming code and $d = n$ for a repetition code.

(3.12) Example.
If r= 3 this means we can put H in standard form like

$$H := \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This gives us as generator matrix:

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

(3.13) Lemma.
Two Hamming codes with the same parameters are equivalent.

Proof. If H and H' are paritity check matrices for Hamming codes than they have the same columns. This means there is a permutation matrix P such that $H' = HP$. This implies that if G' is a generator matrix for the second Hamming code then $G'H'^{\perp} = G'(HP)"ot = (G'P^{\perp})H$ so $G'P^{\perp}$ is a generator matrix for the first Hamming code. Now $P^{\perp}$ is also a permutation matrix so G and G' generate equivalent codes. □

(3.14) Theorem.
$\mathtt{Ham}(r, 2)$ *is a perfect* $[2^r - 1, 2^r - r - 1, 3]$*-code.*

Proof. By the dimension of the parity check matrix we know that $n = 2^r - 1$ and $m = 2^r - 1 - r$. We now have to prove that the minimal weight of a codeword is 3. Suppose thus $\mathtt{c}$ is a codeword with weight one or two. Then we should have that $H\mathtt{c}^t = \mathtt{o}$. But $H\mathtt{c}^t$ is the sum of maximum 2 column vectors of H so this should mean that two columns of H are linearly dependent. This is impossible because two different vectors over $\mathbb{F}_2$ are always linearly independent (*). But the minimal weight is also equal to 3. H contains the columns $\mathtt{100\ldots0}^t$,$\mathtt{010\ldots0}^t$ and $\mathtt{110\ldots0}^t$ at the i, j, k^{th} position. Construct the word $\mathtt{c}$ containing a 1 at those 3 places and zero's everywhere else this is a codeword because

$$cH^t = \mathtt{100\ldots0} + \mathtt{010\ldots0} + \mathtt{110\ldots0} = \mathtt{o}.$$

Finally we prove that $\mathtt{Ham}(r, 2)$ is perfect.

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

□

(3.15) Encoding the Hamming code is done by using the generator matrix. Because $d = 3$ the code is one error correcting and the possible error vectors are of the form $\mathtt{e_j} = \mathtt{0}\cdots\mathtt{010}\cdots\mathtt{0}$ where the one is at position j. The syndrome of $\mathtt{e_j}$ is equal to the j^{th} column of H. To decode we proceed like this:

(3.16) Algorithm. Decoding the binary Hamming code
Suppose we receive the word $\mathtt{r}$ *and G and H are in standard position*

1. *Calculate the syndrome* $\mathtt{s} = H\mathtt{r}^t$
2. *If* $\mathtt{s} = \mathtt{o}$ *the original code word was* $\mathtt{r}$ *and the message words are the first* $2^r - r - 1$ *symbols of* $\mathtt{r}$.

3. *If* $\mathtt{s} \neq \mathtt{o}$ *we suppose that one error has occurred. The position of this error is the position of the column vector of* H *equal to* $\mathtt{s}$.

(3.17) One can generalize binary Hamming codes to Hamming codes over arbitrary fields. However one must take care in the construction of the parity check matrix. One cannot take all possible vectors of $\mathbb{F}_q^r$ to construct the parity check matrix because there are two of them that can be linear dependent of each other and this would destroy the argument (*) in theorem 3.14.
So if we use $v \in F_q^r \setminus \{0\}$ then we cannot use kv where $k \in \mathbb{F}_q \setminus \{0,1\})$. So one has to take for each ray of vectors only one representative in H. The set of all nonzero vectors contains $q^r - 1$ elements and every ray contains $q-1$ elements. Therefore the number of such rays is equal to $\frac{q^r-1}{q-1}$.

(3.18) Example.
If r= 2 and $q = 3$ this means that

$$H := \begin{pmatrix} 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \implies G := \begin{pmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \end{pmatrix}.$$

To compute G we use the fact that $-1 = 2 \mod 3$. So $\mathtt{Ham}(2,3)$ is a [4,2,3]-code.

(3.19) Example.
If r= 2 and $q = 5$ this means that

$$H := \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 4 & 3 & 2 & 1 & 0 & 1 \end{pmatrix} \implies G := \begin{pmatrix} 1 & 0 & 0 & 0 & 4 & 1 \\ 0 & 1 & 0 & 0 & 4 & 2 \\ 0 & 0 & 1 & 0 & 4 & 3 \\ 0 & 0 & 0 & 1 & 4 & 4 \end{pmatrix}.$$

So $\mathtt{Ham}(2,5)$ is a [6,4,3]-code.

(3.20) Example.
If $r = 3$ and $q = 3$ we can put H in standard form like

$$H := \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 1 & 0 \\ 1 & 2 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 \end{pmatrix}$$

and we get a $[13, 10, 3]$-code.

Just like the binary codes we have this theorem

(3.21) Theorem.
$\texttt{Ham}(r, q)$ *is a perfect* $[\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1} - r, 3]$*-code over* $\mathbb{F}_q$

(3.22) Exercise.
Prove the theorem above

(3.23) Exercise.
Design a decoding algorithm for non-binary Hamming codes.

(3.24) Exercise.
Write source code to encode and decode both binary and non-binary Hamming codes.

4 The ternary Golay-code

(3.25) Hamming-codes form an infinite series of codes that are perfect, apart from these codes there are also a limited number of special linear codes. These codes were discovered by Marcel Golay and hence they are called Golay codes.

First we construct the ternary Golay code.

(3.26) Definition.
Consider the field $\mathbb{F}_3$ and take the matrix

$$S_5 := \begin{pmatrix} 0 & 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 \end{pmatrix}.$$

The *ternary Golay code* $\texttt{Gol}(11, 3)$ is a $[11, 6]$ code over $\mathbb{F}_3$ with generator matrix

$$G := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & & & & & & & & & & \\ 0 & & & & & & & & & & \\ 0 & & & \mathbb{I}_5 & & & & & S_5 & & \\ 0 & & & & & & & & & & \\ 0 & & & & & & & & & & \end{pmatrix}$$

and parity check matrix

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

(3.27) Theorem.
The ternary Golay code is a perfect 2 error correcting code.

Proof. One can compute that

$$GG^t = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}.$$

So if $\mathtt{x} := \mathtt{x}_1 \cdots \mathtt{x}_6$ is a message word $\mathtt{x}G$ will be a code word and hence

$$\begin{aligned} \mathtt{x}G \cdot \mathtt{x}G &= \mathtt{x}GG^t\mathtt{x}^t \\ &= 2\left(\sum_{i=2}^{6} \mathtt{x}_\mathtt{i}\right)^2. \end{aligned}$$

Because 1 is the only non zero square in $\mathbb{F}_3$, $\mathtt{x}G \cdot \mathtt{x}G \neq 1$. But in $\mathbb{F}_3$ $\mathtt{x}G \cdot \mathtt{x}G$ is equal to the weight modulo 3. So the weight cannot be $1, 4, 7$ or 10.
The weight of a codeword $\mathtt{x}G$ is always at least the weight of $\mathtt{x}$ because the first 6 symbols of $\mathtt{x}G$ are $\mathtt{x}$.
If $\mathtt{x}$ has weight 1, $\mathtt{x}G$ has weight at least 5 because all rows of G have weights bigger than 4. Suppose that $\mathtt{x}$ has weight 2 then $\mathtt{x}G$ has weight bigger than 4. Take two rows G_i and G_j of G and look at the quotients of the last five entries: $G_{i6}/G_{j6}, \cdots, G_{i11}/G_{j11}$. There are at least 3 different quotients. Every linear combination will have at least 2 nonzero symbols in the last 5 digits, so the weight is at least 4 and hence 5 or more.

If x has weight 3 one can do a similar thing using the fact that the submatrix consisting of the last five digits of 3 rows of G has rank 3.

Now we've proven that the minimal weight is 5, we only have to check the sphere packing identity:

$$\begin{aligned}\sum_{i=0}^{2}\binom{11}{i}2^i &= 1+2\binom{11}{1}+4\binom{11}{2}\\ &= 1+22+220 = 243 = 3^5 = 3^{11-6}\end{aligned}$$

□

(3.28) Aside. The Finnish Football-Pool Connection
The ternary Golay code was first discovered by the Finn Juhani Virtakallio who was determining good strategies for betting on blocks of 11 soccer games. The betting game goes as follows. For every one of the 11 matches one can predict a Win, Lose, or Tie for all 11 games. If you do not miss more than two of them, you win the bet. If a group of players gets together in a pool and makes multiple bets to cover all the options (so that no matter what the outcome, somebody's bet comes within 2 of the actual outcome), then the codewords of the golay code provide a very nice option: the balls around its codewords fill all of the space, with none left over.

It was in this vein that the ternary Golay code was first constructed and its 729 codewords appeared in 1947 in the football magazine Veikkaaja.

Veikkaaja

5 Binary Golay-codes

In this section we describe both the extended binary Golay code and the perfect binary Golay code, because they are both of practical importance.

(3.29) Let B be the 12×12 matrix over $\mathbb{F}_2$

$$B := \begin{pmatrix} 1&1&0&1&1&1&0&0&0&1&0&1\\ 1&0&1&1&1&0&0&0&1&0&1&1\\ 0&1&1&1&0&0&0&1&0&1&1&1\\ 1&1&1&0&0&0&1&0&1&1&0&1\\ 1&1&0&0&0&1&0&1&1&0&1&1\\ 1&0&0&0&1&0&1&1&0&1&1&1\\ 0&0&0&1&0&1&1&0&1&1&1&1\\ 0&0&1&0&1&1&0&1&1&1&0&1\\ 0&1&0&1&1&0&1&1&1&0&0&1\\ 1&0&1&1&0&1&1&1&0&0&0&1\\ 0&1&1&0&1&1&1&0&0&0&1&1\\ 1&1&1&1&1&1&1&1&1&1&1&0 \end{pmatrix}$$

This matrix is symmetric and if one looks at the submatrix consisting op the first 11 rows and columns one can easily check that the rows are cyclic permutations of each other. Another property of B is that $B^2 = \mathbb{I}_{12}$.

(3.30) Definition.
The extended golay-code $\texttt{Gol}(24, 2)$ is a binary code with generator matrix

$$G := [\mathbb{I}_{12}B].$$

Because B is its own inverse, not only $[B\mathbb{I}_{12}]$ is a parity check matrix but G itself as well. Also $[B\mathbb{I}_{12}]$ is a generator matrix for $\texttt{Gol}(24, 2)$.

(3.31) Theorem.
The minimal weight of $\texttt{Gol}(24, 2)$ *is* 8.

Proof. First we prove that the weight of a random codeword is a multiple of 4. Because $GG^t = 0$ the inproduct of two code words is zero. This means that the number of entries where they are both 1 is even. Suppose now that $\mathtt{x}$ and $\mathtt{y}$ are two codewords of which the weight is a multiple of four then

$$w(\mathtt{x} + \mathtt{y}) = w(\mathtt{x}) + w(\mathtt{y}) - 2(\#\text{common } 1's \text{ between } \mathtt{x} \text{ and } \mathtt{y})$$

This expression is again a multiple of four because all its terms are. Notice that the rows of G all have weight 8 or weight 12, so every code word has weight $4k,\ k \in \mathbb{N}$.
Now we prove that no code word can have weight four. Suppose $\mathtt{x}G = \mathtt{x}\mathtt{x}B$ is a code word of weight four. Because $[B\mathbb{I}_{12}]$ is also a generator matrix, there exists a $\mathtt{y}$ such that

$$\mathtt{x}G = \mathtt{x}\,\underbrace{\mathtt{x}B}_{} = \mathtt{y}[B\mathbb{I}_{12}] = \underbrace{\mathtt{y}B}_{}\,\mathtt{y} = \mathtt{x}\mathtt{y}$$

Therefore either $\mathtt{x}$ or $\mathtt{y}$ must have a weight smaller or equal than 2. If this is the case for $\mathtt{x}$ we know that $\mathtt{x}G$ is the sum of at most two rows of G and can never have weight equal to 4. For $w(\mathtt{y}) \leq 2$ we proceed the same. $\square$

(3.32) We will now search for a decoding algorithm for $\mathtt{Gol}(24, 2)$. Because the minimal weight is 8 we will be able to correct all error vectors with weight smaller than 4. Take $H := G$ to be the parity check matrix and suppose we have an error vector $\mathtt{e}$ with weight at most 3. We will split up our error vector in two parts of length 12 $\mathtt{e} := [\mathtt{e}_1, \mathtt{e}_2]$. The syndrome of such and error vector is

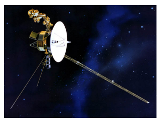
$$\mathtt{s} = G\mathtt{e}^t = \mathtt{e_1}^t + B\mathtt{e_2}^t.$$

Because the weight of $\mathtt{e}$ is smaller than 4 either $\mathtt{e}_1$ or $\mathtt{e}_2$ has weight smaller than 2.
Suppose first that the weight of $\mathtt{e}_2$ is at most 1, then $\mathtt{s}$ the syndrome consist of either a word of weight at most 3 (if $\mathtt{e}_2 = \mathtt{o}$) or a row of B with at most two digits changed.
If the weight of $\mathtt{e}_1$ is at most 1 then one can do the same but now using the syndrome

$$\mathtt{t} = [B\mathbb{I}_{12}]\mathtt{e}^t = B\mathtt{e_1}^t + \mathtt{e_2}^t = B\mathtt{s}.$$

(3.33) Algorithm. Decoding the extended Golay code
We receive the word $\mathtt{r}$.

1. *Compute the syndrome* $\mathtt{s} = G\mathtt{r}^t$
2. *If* $w(\mathtt{s}) \leq 3$ *then* $\mathtt{e} := [\mathtt{s}, \mathtt{o}]$, *stop.*
3. *If* $w(\mathtt{s} + B_i) < 3$ *then* $\mathtt{e} := [\mathtt{s} + B_i, \delta_i]$ *where* δ_i *stands for the vector with everywhere zero's except on the* i^{th} *place a* $\mathtt{1}$. *Stop.*
4. *Compute* $\mathtt{t} := B\mathtt{s}$.
5. *If* $w(\mathtt{t}) < 3$ *then* $\mathtt{e} := [\mathtt{o}, \mathtt{t}]$, *stop.*
6. *If* $w(\mathtt{t} + B_i) < 3$ *then* $\mathtt{e} := [\delta_i, \mathtt{t} + B_i]$, *stop.*
7. *If* $\mathtt{e}$ *is not determined request retransmission.*


Voyager II

(3.34) Aside.
The Voyager mission In the late seventies NASA set up a mission to explore the outer planets of the solar system. This mission visited Jupiter, Saturn, Uranus and Neptune. The images of those planets and their moons had

to be transmitted over several billions of kilometers. In order to achieve good quality NASA used the binary extended Golay code for encoding the photographs.

The last code we will see in this chapter is the binary Golay code.

(3.35) Definition.
The binary Golay code $\texttt{Gol}(23, 2)$ has as generator matrix, the generator matrix of $\texttt{Gol}(24, 2)$ except for the last column which is omitted. Because the minimal weight of $\texttt{Gol}(24, 2)$ is 8 the minimal weight of this code will be 7.

(3.36) Theorem.
$\texttt{Gol}(23, 2)$ *is a perfect* $[23, 12, 7]$*-code.*

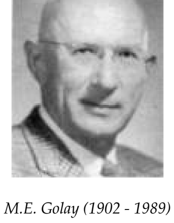
Proof. We only have to calculate the sphere packing identity.

$$\begin{aligned}\sum_{i=0}^{3}\binom{23}{i}2^i &= 1+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}\\ &= 1+23+253+1771 = 2048 = 3^{23-12}\end{aligned}$$

□

(3.37) How do we decode $\texttt{Gol}(23, 2)$? We already have a decoding algorithm for $\texttt{Gol}(24, 2)$ so we can use this. Suppose we receive $\texttt{r}$, we have to transform it into a word of 24 bit. We know that both $\texttt{Gol}(23, 2)$ and $\texttt{Gol}(24, 2)$ can correct 3 errors. Because every error changes the weight of a word by 1 and codewords in $\texttt{Gol}(24, 2)$ have even weight, message words with an odd weight will contain an odd number of errors.
If r contains at most 3 errors we want to add one bit in order to have a message word for $\texttt{Gol}(24, 2)$. We do this by adding a 0 or a 1 such that $w(\texttt{r1})$ or $w(\texttt{r0})$ is odd. Call this new word $\texttt{r}'$. Because $\texttt{r}'$ has odd weight it contains an odd number of errors and we know that it contains at most 3 errors in the first 23 digits, so it contains as a whole also at most 3 errors. Knowing this we can decode r' by extended Golay code algorithm.
In practice, the received word is almost always a code word, however $\texttt{r}'$ is never a codeword. In that case computing the syndrome will give us the last row of $\texttt{B}$. It is useful to check this at the start of the algorithm rather than to wait until step 3.

(3.38) Aside.


M.E. Golay (1902 - 1989)

Marcel J.E. Golay was a Swiss-born mathematician, physicist, and information theorist, who applied mathematics to real-world military and industrial problems. Golay worked on many problems, including the development of the Golay codes. and the generalization of the perfect binary Hamming codes to non-binary codes.

6 Fundamental theorems

(3.39) Have we in fact considered all possible perfect codes or are there other ones? In the case of linear codes this is actually the case, but one can also define perfect non-linear codes. In general van Lint (and others) proved the following theorem

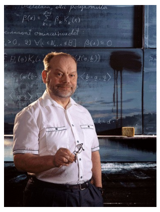
H.J. Van Lint

(3.40) Theorem. van Lint-Tietäväinen
Every perfect linear code over $\mathbb{F}_q$ symbols where q is a prime power, has the parameters of a repetition code, a Hamming or a Golay code ($\mathtt{Gol}(23,2)$ or $\mathtt{Gol}(11,3)$).

(3.41) In 1975 Delsarte and Goethals proved that every code with the parameters of a Golay code is in fact a Golay code. On the other hand one can prove easily that every linear code with the parameters of a Hamming code is a Hamming code.

(3.42) Exercise.
Prove that every linear code with the parameters of a Hamming code is a Hamming code.

(3.43) Exercise.
Prove that every linear code with the parameters of a repetition code is a repetition code.


A. Tietäväinen

(3.44) Theorem. van Lint-Tietäväinen
Every perfect linear code over $\mathbb{F}_q$ is a repetition, a Hamming or a Golay code. ($\mathtt{Gol}(23,2)$ or $\mathtt{Gol}(11,3)$).

(3.45) There exist however perfect non-linear codes that have the parameters of a Hamming code. Those were constructed by Schönheim and Lindström in 1968-69. It remains an open question whether there exist perfect codes over alphabets where the number of symbols is not a prime power.

Chapter 4

Cyclic Codes

1 Introduction

In the previous chapter we saw examples of linear codes, that are practical to use and have good properties, but because these codes are rare one has little flexibility in choosing appropriate parameters for the code you need. In this chapter we will introduce a new variety of codes that are easy to construct and adapt to different situations.

(4.1) Definition.
A linear $[n, m]$-code C is called a *cyclic code* if and only if for every word $\mathtt{x} := \mathtt{x}_1 \cdots \mathtt{x}_n$

$$\mathtt{x}_1 \cdots \mathtt{x}_n \in C \implies \mathtt{x}^{\lhd} := \mathtt{x}_2 \cdots \mathtt{x}_n \mathtt{x}_1 \in C.$$

So every *cyclic shift* of a codeword is again a codeword.

This property seems very strict but in fact there are many codes that are cyclic.

(4.2) Example.
The Hamming code $\mathtt{Ham}(3, 2)$ as we've seen it in the previous chapter is not cyclic but it is equivalent to a cyclic code if we take another parity check and generator matrix. Take

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

One sees that the rows of this matrix are cyclic permutations of each other and if we cycle these rows further they still remain linear combinations of the first 3 rows:

$$\begin{aligned}
\texttt{1001011} &= \texttt{1011100} + \texttt{0010111}\\
\texttt{1100101} &= \texttt{1011100} + \texttt{0101110} + \texttt{0010111}\\
\texttt{1110010} &= \texttt{1011100} + \texttt{0101110}\\
\texttt{0111001} &= \texttt{0010111} + \texttt{0101110}
\end{aligned}$$

So the code with H as generator matrix is cyclic.
Because $\mathtt{x}^{\lhd} \cdot \mathtt{y}^{\lhd} = \mathtt{x} \cdot \mathtt{y}$ the orthogonal complement of a cyclic code is cyclic as well.

Because we introduced an extra property to the linear codes, we can put an extra structure onto the vector space of code words we're working with. With every element of $\mathbb{F}_q^n$ one can associate a polynomial of degree at most $n-1$ like this

$$\mathbb{F}_q^n \to \mathbb{F}_q[X] : \mathtt{c_0} \cdots \mathtt{c_{n-1}} \longmapsto c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}.$$

If we multiply such a polynomial with X we see that all the coefficients shift one position to the right. However because c_{n-1} also shifts one to the right and not to the first position, the new polynomial doesn't correspond anymore to a codeword. This problem is solved by identifying X^n with 1. This means that one has to work in the quotientring

$$\mathbb{F}_q[X]/(X^n - 1)$$

rather than in $\mathbb{F}_q[X]$. In this ring multiplying by X corresponds to a cyclic shift of the coefficients.
We have a bijective map

$$\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q[X] : \mathtt{c_0} \cdots \mathtt{c_{n-1}} \longmapsto [c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}]_{X^n - 1}.$$

and this map satisfies that

$$\mathcal{P}(\mathtt{c}^{\lhd}) = X\mathcal{P}(\mathtt{c}).$$

The image of a cyclic code under $\mathcal{P}$ corresponds to a subset of $\mathbb{F}_q[X]/(X^n - 1)$ closed under addition and multiplication by X and hence multiplication by every element of $\mathbb{F}_q[X]/(X^n - 1)$:

$$(a_0 + a_1 X + \cdots + a_k X^k)\mathcal{P}(\mathtt{c}) = \mathcal{P}(a_0 \mathtt{c} + a_1 \mathtt{c}^{\lhd} + \cdots + a_k \mathtt{c}^{\underbrace{\lhd \cdots \lhd}_{k\times}}$$

Such a subset is called an ideal of $\mathbb{F}_q[X]/(X^n - 1)$.

(4.3) Definition.
An *ideal* of $R = \mathbb{F}_q[X]/(X^n - 1)$ is a subset $\mathfrak{c} \subset R$ such that

$$\forall a, b \in \mathfrak{c} : \forall r \in R : a + b \in \mathfrak{c} \& ra \in \mathfrak{c}$$

(4.4) Theorem.
There is a bijective correspondence between cyclic n-codes over $\mathbb{F}_q$ and ideals in $\mathbb{F}_q[X]/(X^n - 1)$.

Proof. Try it yourself. □

From now on we will identify a cyclic code with its corresponding ideal.

2 Generator polynomial and check polynomial

For linear codes we had a generator matrix, in the case of cyclic codes one can prove the following:

(4.5) Theorem.
For every ideal (or cyclic code) $\mathfrak{c} \subset \mathbb{F}_q[X]/(X^n - 1)$ there exists a polynomial $g(X) \in \mathbb{F}_q[X]/(X^n - 1)$ such that:

$$\mathfrak{c} := \{a(X)g(X) | a(X) \in \mathbb{F}_q[X]/(X^n - 1)\}$$

Proof. Define $g(X)$ to be a polynomial in $\mathfrak{c}$ with lowest degree. Every other polynomial $f(x)\mathfrak{c}$ must be divisible by $g(X)$. If not we could perform a division and write $f(X) = q(X)g(X) + r(x)$ with $\deg r(X) < \deg g(X)$. Because an ideal $\mathfrak{c}$ is closed under addition and multiplication with elements of $\mathbb{F}_q(X)/(X^n - 1)$, $r(x) = f(X) - q(X)g(X)$ is again a code polynomial. This contradicts the fact that $g(X)$ has the lowest degree. □

We call g(X) a *generator polynomial* for $\mathfrak{c}$.
Writing $g(X) = g_0 + g_1X + \cdots + g_kX^k$ we see that $g(X), Xg(X), \ldots, X^{n-k-1}g(X)$ form a $\mathbb{F}_q$-basis for $\mathfrak{c}$ and hence

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_k & & & & \\ & g_0 & g_1 & \cdots & g_k & & & \\ & & \ddots & \ddots & \ldots & \ddots & & \\ & & & & g_0 & g_1 & \cdots & g_k \end{pmatrix} \in \mathrm{Mat}_{n-k \times n}(\mathbb{F}_q)$$

is a generator matrix for the code.

(4.6) Theorem.
For a non-trivial cyclic code $\mathfrak{c} \subset \mathbb{F}_q[X]/(X^n - 1)$ the generator polynomial divides $X^n - 1$.

Proof. If $g(X)$ doesn't divide $X^n - 1$ then one can look at the rest of $X^n - 1$ divided by $g(X)$. This rest $r(X)$ has lower degree than $g(X)$ but it is a linear combination of $g(X)$ and $X^n - 1$ ($= 0 \in \mathfrak{c}$) so it is again an element of the code. But this is impossible because $g(X)$ is the element of the code with the lowest degree. □

(4.7) Definition.
Define $h(X) := \frac{X^n - 1}{g(X)}$ to be the check polynomial of $\mathfrak{c}$.

One can easily prove that $c(X)$ is a code polynomial if and only if

$$h(X)c(X) \equiv 0 \mod X^n - 1.$$

(4.8) If we go back to the vector space representation we can model multiplication by $h(X)$ as matrix multiplication with

$$H = \begin{pmatrix} h_{n-k} & h_{n-k-1} & \dots & h_0 & & & \\ & h_{n-k} & h_{n-k-1} & \dots & h_0 & & \\ & & \ddots & \ddots & \dots & \ddots & \\ & & & h_{n-k} & h_{n-k-1} & \dots & h_0 \end{pmatrix} \in\in \mathrm{Mat}_{k \times n}(\mathbb{F}_q).$$

So this H is the parity check matrix for our code.

(4.9) Example.
Because the parity check matrix of the Hamming code in the previous example

$$H := \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

we can deduce that $h(X) := X^4 + X^2 + X + 1$ and $g(X) := (X^7 - 1)/h(X) = X^3 + X + 1$

(4.10) An interesting way of encoding a cyclic code is systematic encoding. If we encode just by multiplying with $g(X)$ the code word will not contain the original message because the generator matrix

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_k & & & \\ & g_0 & g_1 & \cdots & g_k & & \\ & & \ddots & \ddots & \cdots & \ddots & \\ & & & g_0 & g_1 & \cdots & g_k \end{pmatrix}$$

Does not contain the identity matrix. To remedy this problem one tries to construct a codeword of our code that contains the original message plus some check digits. This is done as follows:
Suppose $u(X)$ is the message we want to encode and that $n-k$ is the degree of $g(X)$. $X^{n-k}u(X)$ will then be the message shifted to the last k of the n digits of the codewords. However $X^{n-k}u(X)$ itself is not a code word. If we want to make a code word without destroying the digits from $u(X)$ one has to change the first $n-k$ digits. Because of the division algorithm we have the identity

$$\exists q(X), r(X): \; X^{n-k}u(X) = q(X)g(X) + r(X).$$

This implies that $X^{n-k}u(X) - r(X)$ is a codeword satisfying our demands, because the two terms of the sum do not overlap.

(4.11) Given any divisor $g(X)|X^n - 1$ we have a uniqe cyclic $[n, m, d]$-code with this divisor as generator polynomial. The dimension of the code is $m = n - \deg g(X)$ but it is not clear what the minimal distance d is for a given $g(X)$ or how one can find a $g(X)$ such that d is a certain given number. In the final chapter we will investigate these problems.

Chapter 5

Reed-Solomon Codes

1 Definition

Reed-Solomon codes are an important sub-class of the cyclic codes. These codes are frequently used in applications. Before we introduce them, we first need to recall the following fact about finite fields.

(5.1) Lemma.
In any finite field $\mathbb{F}_q$ there is an element $\alpha \in \mathbb{F}_q$ such that every nonzero element of the field is a power of α:

$$\mathbb{F}_q = \{0, \alpha^0, \alpha, \alpha^2, \alpha^3, \cdots, \alpha^{q-2}\}$$

This element is called a primitive element.[1]

Note that taking higher powers of α makes you run back through the same elements: $\alpha^{q-1} = \alpha^0 = 1$, $\alpha^q = \alpha^1$, etc. This implies that for every nonzero element α^k we have, $(\alpha^k)^{q-1} = (\alpha^{q-1})^k = 1^k = 1$. Hence, over $\mathbb{F}_q$ the polynomial $X^{q-1} - 1$ can easily be factorized:

$$X^{q-1} - 1 = \prod_{i=0}^{q-2} (X - \alpha^i).$$

(5.2) Definition.
A *Reed-Solomon code* is a cyclic code of length $q-1$ over the field $\mathbb{F}_q$ with generator polynomial

$$g(X) = \prod_{j=0}^{2t-1} (X - \alpha^j)$$

[1]Such an element is not unique. More information about this can be found in the appendix.

Where α is a primitive element of $\mathbb{F}_q$.

(5.3) Theorem.
The parameters of the Reed-Solomon code are $[q-1, q-2t-1, 2t+1]$

Proof. The first parameter is $q-1$ by definition, the second parameter is $q-1-2t$ because $\deg g(X) = 2t$. Now we only need to prove that all codewords have weights bigger than $2t$.
Suppose $e(X) = \sum_{l=1}^{\nu} e_{i_l} X^{i_l}$ is a polynomial of nonzero weight $\nu < 2t+1$ then this can never be a codeword because otherwise $e(X) = g(X)m(X)$ and $e(\alpha^j) = g(\alpha^j)m(\alpha^j) = 0$ for $j \leq 2t-1$, so we would have as equations

$$\begin{aligned} e_{i_1}(\alpha^0)^{i_1} + ... + e_{i_\nu}(\alpha^0)^{i_\nu} &= 0 \\ &\vdots \\ e_{i_1}(\alpha^{\nu-1})^{i_1} + ... + e_{i_\nu}(\alpha^{\nu-1})^{i_\nu} &= 0. \end{aligned}$$

These equations can be solved uniquely as long as the following determinant is not zero.

$$\det \begin{pmatrix} (\alpha^0)^{i_1} & \cdots & (\alpha^0)^{i_\nu} \\ \vdots & & \vdots \\ (\alpha^{\nu-1})^{i_1} & \cdots & (\alpha^{\nu-1})^{i_\nu} \end{pmatrix} = \prod_{\kappa<\lambda} (\alpha^{i_\kappa} - \alpha^{i_\lambda})$$

This identity is obtained by using formula for the Vandermonde determinant[2]. Because the $\alpha^{i_\kappa} \neq \alpha^{i_\lambda}$ whenever $i_\kappa \neq i_\lambda$ and both exponents are smaller than $q-1$ the determinant is not zero and the only possible solution is the zero solution. So a polynomial of nonzero weight smaller then $2t+1$ can never be a codeword.
There is however a codeword of weight $2t+1$ nl. $g(X) = g_0 + g_1 X + \cdots + g_{2t} X^{2t}$. □

(5.4) Theorem.
An $[n, k, d]$ *Reed-Solomon code is maximal distance separable, i.e. it has the greatest possible minimal distance of all possible codes with the same* n *and* k.

Proof. For a general linear $[n, k]$-code $d \leq n-k+1$. This is because the parity check matrix has $n-k$ rows it has rank at most $n-k$ So there is a linear combination of $n-k+1$ columns of H that gives us $\mathtt{0}$. This linear combination can be written as $H\mathtt{x}^t = \mathtt{0}$ for a certain vector $\mathtt{x}$ with weight $n-k+1$. This implies that $\mathtt{x}$ is a codeword and hence $d \leq n-k+1$.

[2] Try to prove this yourself using induction or check `www.proofwiki.org/wiki/Vandermonde_Determinant`

For a Reed-Solomon code $k = q - 2t - 1$ and $d = 2t + 1 = (q - 1) - (q - 2t - 1) + 1 = n - k + 1$ so it has the largest minimal distance possible. $\square$

(5.5) This does not however imply that Reed-Solomon codes are the best codes that exist. There are other combinations of n and k for which there exist no Reed-Solomon codes, but for which other codes exist that have better minimum distance. One of the disadvantages of Reed-Solomon codes is that the size of the alphabet determines the size of the code. There are ways to avoid this problem using more sofisticated techniques in finite fields but we will not go into this.

(5.6) Example.
Let's take $q = 5$ and $t = 1$. A possible choice for α is 2: the powers of 2 mod 5 are:

$$1 = 2^0, 2 = 2^1, 4 = 2^2, 3 = 2^3.$$

This means $g(X) = (X - 1)(X - 2) = X^2 + 2X + 2$ and $\mathtt{RS}(5, 1)$ is a linear $[4, 2, 3]$-code over $\mathbb{F}_5$.

(5.7) Example.
Let's take $q = 7$ and $t = 2$. A possible choice for α is 3: the powers of 2 mod 7 are:

$$1 = 3^0, 3 = 3^1, 2 = 3^2, 6 = 3^3, 4 = 3^4, 5 = 3^5.$$

This means $g(X) = (X - 1)(X - 3)(X - 2)(X - 6)$ and $\mathtt{RS}(7, 2)$ is a linear $[6, 2, 5]$-code over $\mathbb{F}_7$.

(5.8) Aside.
These code were invented in 1960 by Irving S. Reed and Gustave Solomon, who were then members of MIT Lincoln Laboratory. They published their results in an article entitled "Polynomial Codes over Certain Finite Fields." The first application had to wait until 1982 with the invention of the CD. Today Reed-Solomon codes are used in a wide variety of commercial applications, most prominently in CDs, DVDs and Blu-ray Discs

Reed and Solomon

2 Syndromes for Reed Solomon codes

The generator polynomial $g(X)$ for Reed-Solomon codes was chosen specifically such that it has zeros at $1, \alpha, \alpha^2, \ldots, \alpha^{2t-1}$, where α is a primitive element of $\mathbb{F}_q$. This implies that if $w(X)$ is the received polynomial coming from a message $m(X)$ and an error polynomial $e(X)$ we get

$$w(\alpha^i) = m(\alpha^i)g(\alpha^i) + e(\alpha^i) = e(\alpha^i), i = 0, 2, \dots, 2t-1,$$

(5.9) Definition.
For $j = 0, 1, \dots, 2t-1$ we define the *j-th syndrome* for the received polynomial $r(X)$ as

$$S_j = r(\alpha^j) = \sum_{l=1}^{n} e_l(\alpha^j)^l,$$

(5.10) Suppose that a total of ν errors occurred, located at positions $i_1, \dots, i_\nu$ with $\nu \leq t$,

$$e(X) = e_{i_1}X^{i_1} + e_{i_2}X^{i_2} + \cdots + e_{i_\nu}X^{i_\nu}.$$

Then the powers of X define the error locations, and the coefficients determine the error magnitudes. We can now write

$$\begin{aligned} S_0 &= e_{i_1}(\alpha^{i_1})^0 + e_{i_2}(\alpha^{i_2})^0 + \cdots + e_{i_\nu}(\alpha^{i_\nu})^0 \\ S_1 &= e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \cdots + e_{i_\nu}\alpha^{i_\nu} \\ &\vdots \\ S_{2t-1} &= e_{i_1}(\alpha^{i_1})^{2t-1} + e_{i_2}(\alpha^{i_2})^{2t-1} + \cdots + e_{i_\nu}(\alpha^{i_\nu})^{2t-1} \end{aligned}$$

We need to solve this set of equations for the α^{i_l} and the e_{i_l}.

Once these have been found, we can take logarithms (base α) in $\mathbb{F}_q$:

$$\log_\alpha X = j \iff \alpha^j = X \text{ and } 1 \leq j \leq q-1$$

to find the error location numbers i_l, and correct the symbols at these positions by subtracting the corresponding error magnitudes.

(5.11) In order to avoid a profusion of subscripts and superscripts, it is usual to introduce at this point the error location variables $X_l = \alpha^{i_l}$ and the error magnitude variables $Y_l = e_{i_l}$. Using this notation, we have

$$\begin{aligned} S_0 &= Y_1 + Y_2 + \cdots + Y_\nu \\ S_1 &= Y_1X_1 + Y_2X_2 + \cdots + Y_\nu X_\nu \\ &\vdots \\ S_{2t-1} &= Y_1X_1^{2t-1} + Y_2X_2^{2t-1} + \cdots + Y_\nu X_\nu^{2t-1} \end{aligned}$$

This system of $2t$ non-linear equations are symmetric functions of $X_1, X_2, \ldots, X_\nu$ known as power-sum symmetric functions. Any method of solving these equations is a decoding method for BCH codes. Note that if the error location variables are known, the above system of equations is linear in the Y_l, and can be solved using regular techniques from linear algebra. The hard part of the decoding process is to find the error locations.

(5.12) Example. Single error correction
If a single error occurred,

$$\begin{aligned} S_0 &= Y_1 \\ S_1 &= Y_1 X_1 \end{aligned}$$

and hence $X_1 = S_1/S_0$ and $Y_1 = S_0$. Hence if $S_1/S_0 = \alpha^i$, the error is at location i and its value is S_0.

As soon as the number of errors increases we need stronger methods to solve the equations.

3 Finding errors and finding roots

In this section we will describe a very efficient algorithm to decode Reed Solomon codes.

(5.13) Definition.
We define the *syndrome polynomial* as

$$S(X) = \sum_{l=0}^{2t-1} S_l X^l,$$

the *error locator polynomial* as

$$\Lambda(X) = \prod_{l=1}^{\nu} (1 - X_i X)$$

the *error evaluator polynomial* as

$$\Omega(X) = \sum_{k=1}^{\nu} Y_k \prod_{l \neq k} (1 - X_l X)$$

Where the S_i, the $X_l = \alpha^{i_l}$ and the $Y_l = e_{i_l}$ are defined as in the previous section.

(5.14) Note that by definition the roots the error-locator polynomial are directly related to the error location variables:

$$\Lambda(X) = 0 \iff X = \alpha^{-i_l} \text{for some error location } i_l$$

The polynomial itself is however unknown, and must be determined (hopefully in some efficient manner) from the syndrome polynomial. After which we can determine the roots of $\Lambda(X)$ to find the error locations.

(5.15) Although finding the roots of polynomials is in general a hard problem (over the reals there is no general solution for degree 5 of higher), finite field polynomials have the advantage that we can find the roots by exhaustive search over $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{q-1}\}$. Such a search is known as a *Chien search.*

(5.16) After we have found the roots of $\Lambda(X)$ we can determine the sizes of the errors using the error evaluator polynomial. We will first recall formal derivations in $\mathbb{F}_q$ For a polynomial $a(X) = \sum_{i=0}^{s} a_i X^i$ we define

$$a'(X) = \sum_{i=1}^{s} i a_i X^{i-1}$$

Where we take i modulo the characteristic of $\mathbb{F}_q$. For these formal derivatives the same properties, like the Leibniz rule, hold as in the normal case. We can now calculate that

$$\Lambda'(X) = \sum_k -\alpha^{i_k} \prod_{l \neq k} (1 - \alpha^{i_l} X),$$

so by the definition of $\Omega(X)$

$$\Lambda'(\alpha^{-i_k}) = -\alpha^{i_k} \prod_{l \neq k} (1 - \alpha^{i_l - i_k}) = \frac{-\alpha^{i_k}}{e_{i_k}} \Omega(\alpha^{-i_k})$$

This gives us the following expression for the error magnitudes

$$e_{i_k} = \frac{-\alpha^{i_k} \Omega(\alpha^{-i_k})}{\Lambda'(\alpha^{-i_k})}.$$

In view of this discussion, we now need to solve the following problem.

(5.17) Problem.
Given the syndrome polynomial $S(X)$ determine the error locator polynomial $\Lambda(X)$ and the error evaluator polynomial $\Omega(X)$.

4 Berlekamps Algorithm

Consider the ring $\mathbb{F}_q[X]/(X^{2t})$. The polynomial $1-\alpha^j X$ is not divisible by X and hence invertible. Its inverse is computed by chopping of the Taylor expansion at $X^{2t} \equiv 0$

$$\frac{1}{1-\alpha^j X} \equiv \sum_{l=0}^{2t-1} (\alpha^j X)^l \mod X^{2t}.$$

Using this identity we get

$$\begin{aligned} S(X) &= \sum_{l=0}^{2t-1} S_l X^l \\ &= \sum_{l=0}^{n-1} \sum_{k=1}^{\nu} e_{i_k} (\alpha^{i_k} X)^l \\ &= \sum_{k=1}^{\nu} e_{i_k} \left(\sum_{l=0}^{2t-1} (\alpha^{i_k} X)^l \right) \\ &\equiv \sum_{k=1}^{\nu} \frac{e_{i_k}}{1-\alpha^{i_k} X} \mod X^{2t}. \end{aligned}$$

We can relate $\Lambda(X)$, $S(X)$ and $\Omega(X)$ in the following way

$$\Omega(X) = \Lambda(X) \sum_{k=1}^{\nu} \frac{e_{i_k}}{1-\alpha^{i_k} X} \equiv \Lambda(X) S(X) \mod X^{2t}.$$

To find $\Omega(X)$ and $\Lambda(X)$ out of $S(X)$ we will use the algorithm of Euclid. As we saw in the appendix, this algorithm enables us to find the $\gcd$ elements by division.
Suppose thus that $a(X)$ and $b(X)$ are polynomials of $\mathbb{F}_q$ then the algorithm of Euclid supplies us with series $r_i(X)$, $s_i(X)$ and $t_i(X)$ ($i = 1, \dots, \kappa+1$) such that

$$s_i(X)a(X) + t_i(X)b(X) = r_i(X) \text{ and } \deg t_i(X) + \deg r_{i-1}(X) = \deg a(X)$$

With $r_0(X) = a(X)$, $r_\kappa(X) = \gcd(a(X), b(X))$ and $r_{\kappa+1}(X) = 0$.

(5.18) Theorem.
Suppose $t(X)$ and $r(X)$ are nonzero polynomials over $\mathbb{F}_q$ satisfying the following conditions:

1. $\gcd(t(X), r(X)) = 1$,

2. $\deg t(X) + \deg r(X) < \deg a(X)$,

3. $t(X)b(X) = r(X) \mod a(X)$.

Then there exists an index $h \in \mathbb{N}$ and a constant $c \in \mathbb{F}_q$ such that

$$t(X) = ct_h(X) \text{ and } r(X) = cr_h(X).$$

Where the $r_h(X)$ and the $t_h(X)$ are coming from the algorithm of Euclid.

Proof. First observe that the $\deg r_i(X)$ strictly decreases when i increases. By condition 2 we have that $\deg r < \deg a$ and hence there is an index h such that

$$\deg r_h(X) \leq r(X) < \deg r_{h-1}(X).$$

From condition 3 we have that there is an $s(X) \in \mathbb{F}_q[X]$ such that

(1) $s(X)a(X) + t(X)b(X) = r(X)$,

while from Euclid's algorithm we have that

(2) $s_h(X)a(X) + t_h(X)b(X) = r_h(X)$,

Multiplying equation (1) by $t_h(X)$, equation (2) by $t(X)$ and subtracting the two results we obtain

(3) $(t(X)s_h(X) - t_h(X)s(X))a(X) = t(X)r_h(X) - t_h(X)r(X)$,

Now we use the definition of h to get a bound on the degrees. By condition 2 we have

$$\deg t(X)r_h(X) = \deg t(X) + \deg r_h(X) \leq \deg t(X) + \deg r(X) < \deg a(X),$$

and because of Euclids algorithm:

$$\deg t_h(X)r(X) = \deg t_h(X) + \deg r(X) < \deg t_h(X) + \deg r_{h-1}(X) = \deg a(X).$$

So, the right hand side of equation (3) has a degree strictly smaller then $\deg a(X)$, while the left hand side has degree $\geq \deg a(X)$. Therefore both sides must be zero and

(4) $t(X)r_h(X) = t_h(X)r(X)$.

Because $\deg t_h(X) + \deg r_{h-1}(X) = \deg a(X)$, $t_h \neq 0$ and by condition 1 and equation (4), $r(X)$ divides $r_h(X)$ but it has the same or a higher degree so it is a scalar multiple of $r_h(X)$. Dividing the previous equation by $r_h(X)$, we see that $t(X)$ is also a scalar multiple of $t_h(X)$. □

(5.19) Based on this theorem, we can find $\Lambda(X)$ and $\Omega(X)$ because they satisfy the necessary conditions if we identify

$$a(X) := X^{2t},\ b(X) := S(X),\ t(X) := \Lambda(X),\ r(X) := \Omega(X)$$

The error evaluator polynomial has no zeros in common with the error locator polynomial $\Lambda(X)$ because

$$\Omega(\alpha^{-i_k}) = e_{i_k} \prod_{l \neq k} (1 - \alpha^{i_l - i_k}) \neq 0.$$

So $\gcd(\Lambda(X), \Omega(X)) = 1$. The degree of $\Lambda(X)$ is equal to the number of errors $\nu \leq t$. The degree of $\Omega(X)$ is smaller as it is the sum of polynomials of degree $\nu - 1$ and

$$\deg \Lambda(X) + \deg \Omega(X) \leq \nu + \nu - 1 \leq 2t - 1 < \deg X^{2t}$$

(5.20) The constant c must be chosen such that $ct_h(0) = \Lambda(0) = 1$. We claim that h is the unique index such that

$$\deg r_h < t \leq \deg r_{h-1}.$$

Indeed, smaller values of i would result in a polynomial $\Omega(X) = cr_h(X)$ whose degree is larger than $t - 1$. On the other hand we have for every $i > h$

$$\deg t_i \geq \deg t_{h+1} = \deg a - \deg r_h > t$$

So then $\Lambda(X)$ will have a degree larger than t.

If we put every thing together we get the following algorithm

(5.21) Algorithm. ***Berlekamp-Massy-Forney***

1. *Compute the syndrome polynomial $S(X)$ out of the received word $w(X)$.*
2. *Use the algorithm of Euclid for X^{2t} and $S(X)$ to find an index h such that $\deg r_h < t \leq \deg r_{h-1}$.*
3. *Define $\Lambda(X) = t_h(X)/t_h(0)$ and $\Omega(X) = r_h(X)/t_h(0)$. Find the i_k such that α^{-i_k} is a root of $\Lambda(X)$. There have to be $\deg \Lambda(X)$ distinct roots otherwise too many errors have occured and you must ask for retransmission.*
4. *Define*

$$e_{i_k} = \frac{-\alpha^{i_k} \Omega(\alpha^{-i_k})}{\Lambda'(\alpha^{-i_k})}.$$

The corrected polynomial is

$$w(X) - \sum_k e_{i_k} X^{i_k}.$$

(5.22) Aside.

E.E. Berlykamp

Elwyn R. Berlekamp, professor at Berkeley was born in Dover, Ohio on September 6, 1940. In the early 1970s, Dr. Berlekamp founded Cyclotomics, Inc., a research and engineering firm specializing in the development and implementation of high-performance error control systems for digital communications and mass data storage. In 1984, Cyclotomics "Bit-Serial" Reed Solomon encoders were formally adopted as the NASA standard for deep space communications. On the commercial side, all compact disk players use Reed-Solomon Codes with Berlekamp decoding.

Chapter 6

Appendix: Finite fields

Finite fields are a very valuables source of combinatoric constructions, especially in coding theory and cryptography. In this first chapter we will construct these fields and review their properties. This chapter is meant as an overview and hence we only state results without proving them.

1 Prime fields

A field is an object that allows you to work with, like with the ordinary real numbers: you can do addition, substraction, multiplication and division. More formally we state

(6.1) Definition. Field
A field $\mathbb{F}$ is a set equipped with two maps

$$\begin{aligned} + : \mathbb{F}\times\mathbb{F} : (a,b) &\longmapsto a+b \\ \cdot : \mathbb{F}\times\mathbb{F} : (a,b) &\longmapsto ab \end{aligned}$$

such that

`AA` $\forall a,b,c \in \mathbb{F} : (a+b)+c = a+(b+c)$ (associativity of the addition)

`AC` $\forall a,b \in \mathbb{F} : a+b = b+a$ (commutativity of the addition)

`AN` $\exists 0_{\mathbb{F}} \in \mathbb{F} : \forall a \in \mathbb{F} : 0_{\mathbb{F}} + a = a$ (0 is a neutral element for the addition)

`AI` $\forall a \in \mathbb{F} : \exists - a \in \mathbb{F} : a + (-a) = 0_{\mathbb{F}}$ (the addition has inverses)

`MA` $\forall a,b,c \in \mathbb{F} : (ab)c = a(bc)$ (associativity of the multiplication)

`MC` $\forall a,b \in \mathbb{F} : ab = ba$ (commutativity of the multiplication)

MN $\exists 1_{\mathbb{F}} \neq 0_{\mathbb{F}} \in \mathbb{F} : \forall a \in \mathbb{F} : 1_{\mathbb{F}} a = a$ (1 is a neutral element for the multiplication)

MI $\forall a \in \mathbb{F} \setminus \{0_{\mathbb{F}}\} : \exists \frac{1}{a} \in \mathbb{F} : a\frac{1}{a} = 1_{\mathbb{F}}$ (the multiplication has inverses)

D $\forall a, b, c \in \mathbb{F} : (a + b)c = ac + bc$ (distributivity between the addition and multiplication)

If there is no confusion possible we will omit the subscript $\mathbb{F}$ from $0_{\mathbb{F}}$ and $1_{\mathbb{F}}$.

(6.2) The standard examples are the number fields

- $\mathbb{Q}$,
- $\mathbb{R}$,
- $\mathbb{C}$.

The aim of this chapter is now to find examples of fields that contain only a finite number of elements.

(6.3) The starting point of our expedition is the ring of the integers, $\mathbb{Z}$. $\mathbb{Z}$ has most properties of a field, except that there are not inverses for the multiplication. One can solve this problem by introducing fractions leading us to the field of rational numbers $\mathbb{Q}$. However in this way one obtains an infinite field.
There is a second possibility to turn $\mathbb{Z}$ into a field. Although we cannot always divide in $\mathbb{Z}$, there is a division algorithm that given numbers a and p produces a quotient q and a rest r such that

$$a = pq + r, \; 0 \leq r < |p|$$

If we take a fixed p, the trick is now to treat numbers with the same rest divided by p as the same, and define the addition and multiplication up to the rests

(6.4) Definition.
We say that $a, b \in \mathbb{Z}$ are equivalent modulo $p \in \mathbb{N}$ if the differ only by a multiple of p.

$$a \equiv b \mod p \iff p|a - b$$

The equivalence class of a modulo p is the set of integers that are equivalent modulo p with a:

$$[a]_p := \{b \in \mathbb{Z} | a \equiv b \mod p\}$$

The ring $\mathbb{Z}_p := \{[a]_p | a \in \mathbb{Z}\}$ is the ring of rest classes modulo p together with the obvious addition and multiplication.

$$[a]_p + [b]_p := [a+b]_p,\ [a]_p \cdot [b]_p := [a \cdot b]_p.$$

For sake of simplicity we will abandon the notation $[a]_p$ in favor of a when it is obvious what we mean.

One can check that for every number in $\{0, \dots, p-1\}$ there is a unique equivalence class so one can identify the classes with those numbers. Addition and multiplication are a piece of cake: one computes it in the ring of integers and then one takes the rest for division by p.
Computing inverses for the multiplication is more complicated. $[b]_p$ is the inverse of $[a]_p$ if

$$[a]_p[b]_p = [1]_p \text{ or } \exists q \in Z : ab + pq = 1.$$

For random a and p such a b can only exists if their greatest common divisor is 1 and in that case one can compute this b by using the algorithm of Euclid.

(6.5) Algorithm. The algorithm of Euclid
Suppose a and b two integers, we compute their gcd and write it as a linear combination of a and b.

1. *Set r_0 the biggest of a and b in absolute value, and r_1 the other one. Put $i = 1$ and define*

$$\begin{aligned} t_0 &= 1, & t_1 &= 0 \\ s_0 &= 0, & s_1 &= 1 \end{aligned}$$

2. *If $r_i \neq 0$ divide r_{i-1} by r_i and call the quotient q_i and the rest r_{i+1}. Put also*

$$\begin{aligned} t_{i+1} &= -q_i t_i + t_{i-1} \\ s_{i+1} &= -q_i s_i + s_{i-1} \end{aligned}$$

increment i by 1 and repeat this until $r_i = 0$. If this is the case r_{i-1} is the greatest common divisor.

3. *Because of the definition of the s and t we have for each j the identity*

$$r_j = t_j a + s_j b$$

Especially for $j = i - 1$ we have expressed the greatest common divisor in terms of the original polynomials.

Because inverses only exist if the gcd is 1, $\mathbb{Z}_p$ will only be a field if p is a prime in this case every $a \neq [0]_p$ will have $gcd(a, p) = 1$.

(6.6) Theorem.
$\mathbb{Z}_p$ is a field if and only if p is a prime, in that case we also denote is by $\mathbb{F}_p$.

(6.7) Exercise.
Write source code that enable you to add, subtract, multiply and divide in $\mathbb{F}_p$ for a random prime p.

2 Polynomial rings

In the previous section we have already constructed a infinite number of finite fields, but there are more. In order to find the others we will introduce here polynomial rings over fields.

(6.8) Definition.
The ring of polynomials over a field $\mathbb{F}$ is

$$\mathbb{F}[X] := \{\sum_{i=0}^{n} a_i X^i | 0 \leq n < \infty, a_i \in \mathbb{F}\}$$

Addition and multiplication are the usual like in $\mathbb{R}[X]$. The degree of a polynomial is the highest coefficient

(6.9) Example.
Adding polynomials over $\mathbb{F}_2$ is like adding normal polynomials keeping in mind that $1 + 1 = 0$ and hence

$$\begin{aligned}(X^5 + X^3 + X + 1) + (X^4 + X^3 + 1) &= X^5 + X^4 + (1 + 1)X^3 + X + (1 + 1) \\ &= X^5 + X^4 + X.\end{aligned}$$

The product of two polynomials is also obvious

$$\begin{aligned}(X^2 + 1)(X^3 + X + 1) &= (X^5 + X^3 + X^2) + (X^3 + X + 1) \\ &= X^5 + X^2 + X + 1\end{aligned}$$

This structure is not a field because we still don't have inverses but we have overcome the problem of the zero devisors because the degree (the highest non-zero power of X) of the product of two polynomials is the sum of the

degrees of those two polynomials. Nevertheless it is an important structure in the way that all possible finite fields are derived from such rings.
Although we don't have inverses in $\mathbb{F}[X]$, we can divide polynomials with the division algorithm. Given two polynomials $a(X), b(X)$ we can compute the quotient $q(X)$ and rest $r(X)$ such that

$$a(X) = b(X)q(X) + r(X) \text{ and } \deg r(X) < \deg b(X)$$

f.i. in $\mathbb{F}_2$

$$X^4 + X^2 + X + 1 = (X^2 + X + 1)(X^2 + X + 1) + X$$

Just as in normal division theory with the natural numbers, we can define prime (or more correct irreducible) polynomials which have no nontrivial divisors. Like with ordinary numbers we can decompose every polynomial in his prime components.

$$X^4 + X^2 + X + 1 = (X + 1)(X^3 + X^2 + 1), (\text{ in } \mathbb{F}_2).$$

We also can define te concept op greatest common divisor (gcd) and least common multiple (lcm) of two or more polynomials. An important property of the gcd is that we can express $gcd(a(X), b(X))$ as a combination of multiples of $a(X)$ and $b(X)$. For this we again use the algorithm of Euclid, adapted to polynomials. The lcm on the other hand can be calculated by dividing the product of the two polynomials by its gcd.

3 Quotient rings

As we have seen a polynomial ring behaves more or less like the ring of integers, $\mathbb{Z}$. Therefore it can be used in the same way to construct new finite fields.
Consider a polynomial $g(X)$ of degree n in $\mathbb{F}_p[X]$ Two polynomials $a(X)$ and $b(X)$ are said to be equivalent modulo $g(X)$ if they have the same rest if divided by $g(X)$ or equivalently if their difference divides $g(X)$. We write

$$a(X) = b(X) \mod g(X)$$

We construct a new ring by looking again only up to equivalence modulo $g(X)$. We consider a new ring $\mathbb{F}_2[X]/(g)$ which consists of elements

$$[a(X)]_g := \{b(X) \in \mathbb{F}_2[X] | a(X) = b(X) \mod g(X)\}$$

the addition and multiplication are defined as

$$[a(X)]_g + [b(X)]_g = [a(X) + b(X)]_g,\ [a(X)]_g \cdot [b(X)]_g = [a(X)b(X)]_g$$

Because rests always have a lower degree than g, every polynomial is equivalent with one of degree smaller than n. Therefore there are only p^n elements in this ring corresponding to all possible rests. The easiest way to work in this ring is to work only with the rests and each time you multiply you must calculate the rest of the product divided by g.
It is not necessarely true that this ring has no zero divisors because we could have the situation that the product of two rests gives us a multiple of $g(X)$ f.i.

$$[X^2]_{X^4}[X^3]_{X^4} = [X^5]_{X^4} = [0]_{X^4}$$

This is only possible when $g(X)$ is not a prime polynomial because if $g(X)$ were prime and $g(X)|a(X)b(X)$ then $a(X)$ or $b(X)$ must contain $g(X)$ in its prime decomposition and thus its rest will be zero. Computing inverses is also done by using the algorithm of Euclid for polynomials.
For sake of simplicity we will drop the notation $[\cdot]_g$ and denote $[X]_g$ by a greek letter f.i. ξ. We easily have that $[a(X)]_g := a(\xi)$.

(6.10) Example.
If we take $g(X) := X^2 + X + 1$ the field $\mathbb{F}_2[X]/(g(X))$ will consist of four elements: 0, 1, ξ, $\xi + 1$ with the following tables for multiplication and addition:

$+$	0	1	ξ	$\xi+1$
0	0	1	ξ	$\xi+1$
1	1	0	$\xi+1$	ξ
ξ	ξ	$\xi+1$	0	1
$\xi+1$	$\xi+1$	ξ	1	0

$\cdot$	1	ξ	$\xi+1$
1	1	ξ	$\xi+1$
ξ	ξ	$\xi+1$	1
$\xi+1$	$\xi+1$	1	ξ

(6.11) Example.
If we take $g(X) := X^2+1$ then $\mathbb{F}_2[X]/(g(X))$ is not a field anymore because $X^2 + 1 = (X + 1)^2$. It will consist again of four elements: $0, 1, \xi, \xi + 1$, but $\xi + 1$ won't be invertible because $(\xi + 1)^2 = 0$.

$\cdot$	1	ξ	$\xi+1$
1	1	ξ	$\xi+1$
ξ	ξ	1	$\xi+1$
$\xi+1$	$\xi+1$	$\xi+1$	0

The number of elements in a finite field will be exactly p^n with n the degree of the prime polynomial. For $p = 2$, $n = 2$ we just have one irreducible polynomial: $X^2 + X + 1$. For n bigger this is not true anymore. If n is 3 we have exactly 2 irreducibles:

$$X^3+X+1,\ X^3+X^2+1.$$

This does not mean that there are two different kinds of fields with 8 elements. Mathematically $\mathbb{F}_2[X]/(X^3+X+1)$ and $\mathbb{F}_2[X]/(X^3+X^2+1)$ are isomorphic. This means that we can identify the elements of both fields. To put it more clearly take $\xi := [X]_{X^3+X+1}$ and $\eta := [X]_{X^3+X^2+1}$ than we see that

$$\begin{aligned}(\xi+1)^3+(\xi+1)^2+1 &= \xi^3+\xi^2+\xi+1+\xi^2+1+1\\ &= \xi^3+\xi+1=0\end{aligned}$$

So $\xi+1$ fulfills the same equation in the first field as η in the second field. Via this method we can identify all the elements of the two fields with each other:

$$\begin{array}{ccc} 0 & \longmapsto & 0\\ 1 & \longmapsto & 1\\ \xi & \longmapsto & \eta+1\\ \xi+1 & \longmapsto & \eta \end{array}\qquad\qquad \begin{array}{ccc} \xi^2 & \longmapsto & \eta^2+1\\ \xi^2+1 & \longmapsto & \eta^2\\ \xi^2+\xi & \longmapsto & \eta^2+\eta\\ \xi^2+\xi+1 & \longmapsto & \eta^2+\eta+1 \end{array}$$

One can prove that for every n there exists at least 1 irreducible polynomial of degree n and that if there exists more of them they all induce isomorphic fields.

(6.12) Theorem.
For every prime power $q := p^n$ there exists a unique finite field with q elements, this is called the Galois field with q elements and is denoted by $\mathbb{F}_q$.

E. Galois (1811-32)

(6.13) Aside.
Evariste Galois was a mathematican, born in Bourg-la-Reine, France. He was educated privately and at the Collège Royal de Louis-le-Grand. Despite mathematical ability he failed the entrance for the Ecole Polytechnique to study maths, and settled for the Ecole normale Supérieure in 1829 to train as a teacher, but was expelled in 1830 for republican sympathies. He engaged in political agitation, was imprisoned twice, and was killed in a duel aged 21. His mathematical reputation rests on original genius in the branch of higher algebra known as group theory.

4 Galois fields

We will now investigate some properties of those fields. In what follows we will set q equal to p^n.

(6.14) Theorem.
for a finite field $\mathbb{F}_q$, $\mathbb{F}_q^* := \mathbb{F}_q \setminus \{0\}$ will be a commutative cyclic group with $q-1$ elements.

This theorem states that there exists an element $\alpha \in \mathbb{F}_q^*$ such that every other element of $\mathbb{F}_q^*$ can be expressed as a power of α. For example in $\mathbb{F}_8$ we can take ξ itself:

$$\begin{array}{rcl} [0 & = & 0] \\ 1 & = & \xi^0 \\ \xi & = & \xi^1 \\ \xi & = & \xi^3 \end{array} \qquad \begin{array}{rcl} \xi^2 & = & \xi^2 \\ \xi^2+1 & = & \xi^6 \\ \xi^2+\xi & = & \xi^4 \\ \xi^2+\xi+1 & = & \xi^5 \end{array}$$

Here we used all the powers of ξ until the sixth, if we compute the seventh power we will see that its again 1 so we have for every element in $\mathbb{F}_8^*$ an infinite number of possibilities to express it as a power of ξ:

$$\xi + 1 = \xi^3 = \xi^{10} = \xi^{17} = \ldots$$

One can also take η because $\xi^3 = \eta$ thus $\xi = \xi^{15} = \eta^5$ and so f.i.

$$\xi^2 + \xi + 1 = \xi^5 = \eta^{25} = \eta^4.$$

However it is not true that one can always take the generator element of the galois field. Consider $\mathbb{F}_{16} \cong \mathbb{F}_2[X]/(X^4 + X^3 + X^2 + X + 1)$ and take ξ to be the equivalence class of X. One can compute that

$$\begin{aligned} \xi^5 &= \xi(\xi^3 + \xi^2 + \xi + 1) \\ &= (\xi^3 + \xi^2 + \xi + 1) + \xi^3 + \xi^2 + \xi \\ &= 1. \end{aligned}$$

In general there is no algorithm to find a generator of the cyclic group, so one has to do some trial and error.
Consider an element $\alpha \in \mathbb{F}_q$, with $q = 2^n$. A polynomial $a(X) \in \mathbb{F}_2[X]$ such that $a(\alpha) = 0$ is called a *characteristic polynomial* for α, the characteristic polynomial of least degree is called the *minimal polynomial* of α. One can show that the minimal polynomial divides every other characteristic polynomial and that it is irreducible. F.i. In $\mathbb{F}_4$ $s(X) = X^3 + 1$ is a characteristic polynomial for ξ because $\xi^3 = (\xi + 1)\xi = \xi + 1 + \xi = 1$ but it is not its minimal polynomial since

$$X^3 + 1 = (X + 1)(X^2 + X + 1) \text{ and } \xi^2 + \xi + 1 = 0.$$

The degree of an element of a finite field is defined as the degree of its minimal polynomial. Not all the elements of a finite field have the same degree. In $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4+X+1)$ ξ obviously has degree 4 but for ξ^5 we have that

$$\xi^{10} = \xi^2(\xi+1)^2 = \xi^4+\xi^2 = \xi+1+\xi^2 = \xi(\xi+1)+1 = \xi^5+1.$$

So ξ^5 has minimal polynomial X^2+X+1. This implies also that $\mathbb{F}_{16}$ contains a subfield generated by ξ^5. This subfield contains the 4 elements 0, 1, ξ^5 and ξ^5+1 and it is isomorphic to $\mathbb{F}_4$. Although $\mathbb{F}_{16}$ contains $\mathbb{F}_4$ it does not contain $\mathbb{F}_8$. This is because if it would there would exist an element α such that $\alpha^7 = 1$ suppose that this element can be written as ξ^j then $\xi^{7j} = 1 = \xi^{15}$ so 15 must divide $7j$ and thus also $15|j$ and $\alpha = 1$ which is a contradiction.

(6.15) Theorem.
Generally one can embed $\mathbb{F}_{p^m}$ in $\mathbb{F}_{p^n}$ if and only if $m|n$, moreover there is only one subfield of $\mathbb{F}_{p^n}$ isomorphic to $\mathbb{F}_{p^m}$. If ξ is a generator of $\mathbb{F}^*_{p^n}$ and then $\mathbb{F}^*_{p^m}$ will be generated by

$$\xi^{\frac{p^n-1}{p^m-1}}.$$

Some elements of a finite field have the same minimal polynomial, in $\mathbb{F}_4$ e.g. ξ and $\xi+1$ both satisfy X^2+X+1, ξ by definition, $\xi+1$ because

$$(\xi+1)^2+\xi+1+1 = \xi^2+1+\xi+1 = \xi+1+1+\xi+1.$$

One can prove that every minimal polynomial of an element in a finite field has as many roots as it's degree. In $\mathbb{F}_8$ for example the minimal polynomial of ξ is X^3+X+1, but one can compute also that but ξ^2 and ξ^4 are roots of the same polynomial.

$$(\xi^2)^3+\xi^2+1 = (\xi+1)^2+\xi^2+1 = \xi^2+1+\xi^2+1 = 0$$
$$(\xi^4)^3+\xi^4+1 = (\xi+1)^4+\xi^4+1 = \xi^4+1+\xi^4+1 = 0.$$

this is also a general rule if α is a root of a polynomial then $\alpha^p, \alpha^{p^2}, \alpha^{p^3}, \ldots$ will be also roots. Take care some, of those roots will be the same as previous ones. In $\mathbb{F}_8$ We have that $\xi^8 = \xi$, so there are not any new roots apart from ξ, ξ^2 and ξ^4.
Because $\mathbb{F}^*_q$ is a cyclic group of $q-1$ elements, we have that $\alpha \in \mathbb{F}^*_q$

$$\forall \alpha \in \mathbb{F}^*_q : \alpha^{q-1} = 1$$

This implies that every element of $\mathbb{F}_q$, including the zero is a root of the polynomial $X^q - X$. So all the minimal polynomials divide $X^q + X$ so that $X^q + X$ is the product of all minimal polynomials of elements in $\mathbb{F}_q$.

$$X^9 - X = \underbrace{X}_{0}\underbrace{(X+1)}_{1}\underbrace{(X^3+X+1)}_{\xi,\xi^2,\xi^4}\underbrace{(X^3+X^2+1)}_{\xi^3,\xi^5,\xi^6}$$

As we considered polynomials over the prime field $\mathbb{F}_p$, we can also consider polynomials over the field $\mathbb{F}_q$. Most of the properties of these polynomials are the same as those of $\mathbb{F}_p[X]$. We again have a division algorithm and one can also define irreducible polynomials, and we have a unique factorization theorem. Notify however that a polynomial which is irreducible over $\mathbb{F}_2$ might not be irreducible for a bigger field. In $\mathbb{F}_4$

$$X^2 + X + 1 = (X + \xi)(X + \xi + 1).$$

In general when α is a root of a polynomial $a(X)$ over a field then $X - \alpha$ divides $a(X)$. To calculate the quotient $a(X)/(X + \alpha)$ one uses the method of Horner.

(6.16) Example.
Compute the quotient of $f(X) := X^4 + \xi X^3 + X^2 + (\xi + 1)X + \xi$ divided by $X - \xi$ over $\mathbb{F}_4$

	1	ξ	1	$\xi + 1$	ξ
$+$		ξ	0	ξ	ξ
$=$	1	0	1	1	0

In the upper line we put the coefficients of $f(X)$. The third line is the sum of the two upper lines and the k^{th} element of the second line is ξ times the $(k-1)^{th}$ element of the third line. The coefficients of the quotient are all but the last element of the third line, so here the quotient is $X^3 + X + 1$.

One could also try to produce fields taking polynomials over $\mathbb{F}_q$ but those fields will be isomorphic to the one we already found. For instance if we take the polynomial $X^2 + X + \xi$ over $\mathbb{F}_4$ and represent $\alpha = [X]_{X^2+X+\xi}$ than we can identify the new field with $\mathbb{F}_{16}$ because

$$\begin{aligned}
\alpha^4 &= (\alpha + \xi)^2 \\
&= \alpha^2 + \xi^2 \\
&= \alpha^2 + \xi + 1 \\
&= \alpha^2 + \alpha^2 + \alpha + 1 \\
&= \alpha + 1
\end{aligned}$$

So α satisfies the polynomial $X^4 + X + 1$ over $\mathbb{F}_2$ and so we can identify it with an element of $\mathbb{F}_{16}$.

(6.17) Exercise.

- Write source code that can add, multiply and divide over a finite field $\mathbb{F}_p[X]/(g(X))$ if p and $g(X)$ are given.
- Write source code that can find a generator element of $\mathbb{F}_p[X]/(g(X))^*, \cdot$ and produces a conversion table between the additive and the exponential notation of the finite field.
- Write source code that generates prime polynomials over $\mathbb{F}_p[X]$

Index

Contents