Introduction to Coding

for EE511

by

A. DAHIMENE

Institut de Génie Electrique et Electronique

Université M'Hamed Bougara
Boumerdes

Table of content

Mathematical Review

Algebraic structures

Group: It consists of a set of elements **G** = {a, b, c, …} together with a composition law:

$\mathbf{G}\times\mathbf{G} \rightarrow \mathbf{G}$

$(a, b) \rightarrow c = a \triangle b$

In many cases, the operation $\triangle$ is denoted either as "+" or ".". The composition law will be denoted in the rest as "+". It has the following properties:

1. **Closure:** $\forall a, b \in \mathbf{G}, \exists\, c \in \mathbf{G}, c = a + b$
2. **Associativity:** $\forall a, b, c \in \mathbf{G}, a + (b + c) = (a + b) + c$
3. **Neutral element:** $\exists\, e \in \mathbf{G}$,e unique, such that $\forall a \in \mathbf{G}, a + e = e + a = a$

   e is denoted "0" for addition, "1" for multiplication
4. **Inverse:** $\forall a \in \mathbf{G}, \exists a\_ \in \mathbf{G}$ such that $a + a\_ = a\_ + a = e$
5. If the operation is **commutative**, the group is called **abelian.**

   $\forall a, b \in \mathbf{G}, a + b = b + a$

Examples:

Real numbers with arithmetic addition is an abelian group ($\mathbb{R}$ **,** +).

$\mathbb{R}^* = \mathbb{R} - \{0\}$ is an abelian group with multiplication ($\mathbb{R}^*$, .).

The set of natural numbers (positive integers) $\mathbb{N}$ along with addition does not form a group but the set of integers $\mathbb{Z}$ (signed integers) is a group ($\mathbb{Z}$, +).

The set of even integers is a subgroup of ($\mathbb{Z}$, +).

Subgroup: (**S**, +) is a subgroup of (**G**, +) if $\mathbf{S} \subset \mathbf{G}$ and (**S**, +) is itself a group.

Isomorphism: if there exists a one to one mapping $\boldsymbol{f}$ between two sets $\mathbf{G_1}$ and $\mathbf{G_2}$ while ($\mathbf{G_1}$, $\triangle$) and ($\mathbf{G_2}$, $\otimes$) form each one some algebraic structure with:

$a_1, a_2 \in \mathbf{G_1}, b_1, b_2 \in \mathbf{G_2}, \boldsymbol{f}(a_1 \triangle a_2) = b_1 \otimes b_2$ then $\boldsymbol{f}$ is called an isomorphism.

One dimensional translation by fixed increment in one direction is isomorphic with $\mathbb{N}$. One dimensional translation by fixed increment in both directions is isomorphic with $\mathbb{Z}$.

The above examples have an infinite number of elements. Groups with a finite number of elements exist.

Example: Addition modulo an integer.

Consider the set {0, 1, 2}; it is closed under the following addition table:

+	**0**	**1**	**2**
0	0	1	2
1	1	2	0
2	2	0	1

Consider now the set {0, 1, 2, 3}; it is closed under the following addition table:

+	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

The N roots of 1, $a_k = \exp(\frac{j2\pi k}{N})$, k = 0, 1,…, N-1 form a finite group under multiplication: $a_k\, a_j = a_{(k+j) \bmod N}$. The group is cyclic.

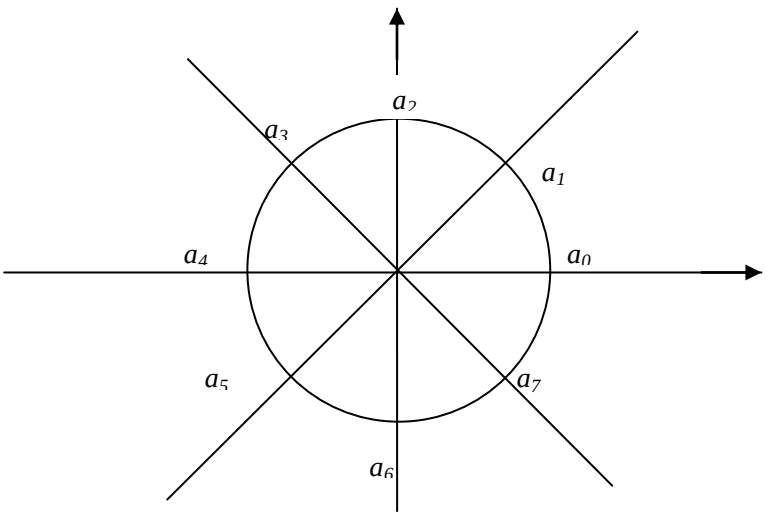


Figure 1 Roots of one

The above figure shows the location of the roots for N = 8.

We can remark that $a_0^n = a_0$ for any value of n. However, a_1 is different:

$a_1^0 = 1 = a_0$; $a_1^1 = a_1$; $a_1^2 = a_2$; $a_1^3 = a_3$; ...; $a_1^7 = a_7$; $a_1^8 = 1$.

Successive powers of a_1 generate all the elements of the group. We say that a_1 is of order 8. a_1 is an element α such that $\alpha^N = 1$. We say that α is the "generator" of the group. a_1 is also called "primitive element" since it generates the entire group. If we repeat the same power raising operation with a_2, we find that it generates a subgroup with 4 elements. a_2 is of order 4 and is not primitive.

Ring: A ring consists of a set **R** = {*a*, *b*, …} closed under two composition operations which we call "+" and "." With the following properties:

1. It is an abelian group under "+".
2. The second operation is associative: $a(bc) = (ab)c$
3. Distributivity: the multiplication distributes over the addition.

$$a\,(b + c) = ab + ac$$

$$(b + c)\,a = ba + ca$$

The set of integers forms a ring ($\mathbb{Z}$, +, .).

If the multiplication is commutative, we say that the ring is commutative.

The set of (nxn) matrices forms a non-commutative ring since **AB** ≠ **BA** in general. There can exist a neutral element for the multiplication called multiplicative identity. For example, 1 for integers,

$$\mathrm{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & \dots & . \\ 0 & 0 & \dots & 1 \end{pmatrix} \text{ for (nxn) matrices}$$

If there is an identity element, then some elements can have an inverse. The element 0 (additive neutral element) does not have an inverse while for (nxn) matrices, only the non singular ones can have inverses.

Important example:

The set of integers {0, 1, …, m-1} with addition modulo m and multiplication modulo m forms a commutative ring with identity.

Consider m = 6; the set is: {0, 1, 2, 3, 4, 5} with the addition table:

+	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

We can remark that -1=5, -2=4, -3=3, -4=2, -5=1.

The multiplication table is

.	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

We can remark that many elements do not have a multiplicative inverse:

2^{-1} does not exist, $5^{-1} = 5$. If we consider the set {1, 2, 3, 4, 5}, we can remark that there is no primitive element.

If n = 2, the set is {0, 1} with the following tables:

+	**0**	**1**
0	0	1
1	1	0

.	**0**	**1**
0	0	0
1	0	1

We can remark that 1 is a primitive element of the set {1}, the case n = 3 is however more informative. The set is {0, 1, 2} with the following multiplication table:

.	**0**	**1**	**2**
0	0	0	0
1	0	1	2
2	0	2	1

Every element of {1, 2} = {0, 1, 2} – {0} has an inverse: $2^{-1} = 2$ and
$2^0 = 1$, $2^1 = 2$, $2^2 = 1$, so 2 is of order 2 and is primitive. So the considered set can be written as $\{0, 2^0, 2^1\}$
Consider now the case n = 5, the set is {0, 1, 2, 3, 4} and the multiplication table is:

.	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

In this case also every non-zero element has an inverse:
$1^{-1} = 1$, $2^{-1} = 3$, $3^{-1} = 2$, $4^{-1} = 4$.
Show that the elements 2 and 3 are of order 4 and are both primitive while 4 is of order 2 and is not primitive.

Ring of polynomials

A polynomial is a sequence of numbers from some algebraic structure (ring in general) where x is called the indeterminate and is used to indicate the position of the actual coefficient in the sequence.

$p(x) = a_0 + a_1 x + a_2x^2 + \ldots + a_nx^n$

If $a_n \neq 0$, we say that $p(x)$ is a polynomial of degree n.

Addition of polynomials

It is performed component wise.

Multiplication of two polynomials

Let $a(x) = a_0 + a_1 x + a_2x^2 + \ldots + a_nx^n$ and $b(x) = b_0 + b_1 x + b_2x^2 + \ldots + b_nx^n$ and

$c(x) = a(x)b(x) = c_0 + c_1 x + c_2x^2 + \ldots + c_{n+m}x^{n+m}$

The coefficients c_i are computed using the usual convolution operation:

$c_0 = a_0b_0$

$c_1 = a_0b_1 + a_1b_0$

$c_2 = a_0b_2 + a_1b_1 + a_2b_0$ ....

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

In the above summation, it is understood that $a_i = 0$ for $i>n$ and $b_i = 0$ for $i<0$ and $i>m$.

If the set of coefficients is the set of integers, it is easy to show that the set of polynomials over $\mathbb{Z}$ forms a ring that contains an infinite number of elements.

Field:

A field is a structure (**F**, +, .) such that:

(**F**, +) is an abelian group and (**F** – {0}, .) is also an abelian group along with the property of distributivity of "." over "+":

$a(b+c) = ab + ac$

Examples of fields:

1) The set of rational numbers $\mathbb{Q}$ is a field with a countable infinite number of elements.

2) The set of real numbers $\mathbb{R}$ on the other hand is a field with an uncountable infinite number of elements.

3) Consider an integer $q \in \mathbb{N}$. If q is prime, the set of integers $\{0, 1, \ldots, q\text{-}1\}$ with addition and multiplication modulo q is a field with a finite number of elements. This field is called a "prime Galois field GF(q)". In the case of finite fields, we can introduce the notion of field characteristic.

If in a given field, there exist a positive integer k such that $\sum_{i=1}^{k} 1 = 0$, the minimum positive value of k is called the field characteristic. Fields that contain an infinite number of elements are said to have a characteristic of zero.

A return to polynomials

Factorization and roots of polynomials

Euclidian division:

As for integers, we can define Euclidian division for polynomials.

Let us consider the set F[x] of polynomials with coefficients taken from a field F, and two polynomials $a(x)$ and $b(x) \in$ F[x] of degree respectively n and m. There exist two unique polynomials $q(x)$ and $r(x)$ such that:

$$a(x) = b(x)q(x) + r(x)$$

and $$\deg[r(x)] < \deg[b(x)]$$

$a(x)$ is called the dividend, $b(x)$ the divisor, $q(x)$ is the quotient and $r(x)$ is the remainder.

Example:

In $\mathbb{R}[x]$, $a(x) = x^3 + 2x + 1$, $b(x) = x^2 + 1$ give $q(x) = x$ and $r(x) = x + 1$. We can write:

$$x^3 + 2x + 1 = (x^2 + 1)x + x + 1$$

Roots and factors

If the divisor is $x - \alpha$, the remainder will be of degree zero and it will be a scalar from the field F. At that time, the Euclidian division becomes:

$$a(x) = (x - \alpha)q(x) + a(\alpha)$$

$a(\alpha)$ is the remainder and it is the evaluation of the polynomial $a(x)$ at α. We can write it as:

$$a(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_n\alpha^n$$

If this remainder is zero, we say that α is a root of the polynomial $a(x)$. If we divide the polynomial $a(x)$ by an arbitrary polynomial $b(x)$ and the remainder is zero, then we can write:

$$a(x) = b(x)q(x)$$

The two polynomials $q(x)$ and $b(x)$ are factors of $a(x)$. If a polynomial cannot be factorized, we say that it is irreducible.

Example: $x^2 + 1$ is irreducible in $\mathbb{R}[x]$.Field of polynomials

We have seen that the modulo operation has allowed us to define finite fields. In this part of the course, we are going to use the modulo operation to define fields of polynomials. In our definition of polynomial multiplication (see p.7), the product of two polynomials of degree m and n respectively produces a polynomial of degree $m + n$. However, as we did with the finite groups, rings and fields of integer, we are going to introduce a multiplication modulo some polynomial.

Given a polynomial $p(x)$ of degree n and any polynomial $a(x)$, the polynomial $a(x)\text{mod}[p(x)]$ is the remainder of $a(x)$ when it is divided by $p(x)$. The polynomial $a(x)\text{mod}[p(x)]$ is a polynomial with a maximum degree of $n - 1$.

Let us now consider the set of polynomial over some field with a maximum degree of $n - 1$. Let us define also a polynomial $p(x)$ of degree n. This set is closed under addition (and it forms an abelian group). It is also closed under multiplication modulo $p(x)$. In general, such set forms a commutative ring with identity. The zero polynomial is a polynomial with all coefficients being zero. The multiplicative identity is a polynomial with all coefficients being zero except the degree zero coefficient being the multiplicative identity of the field. Some polynomials admit multiplicative inverses. There can exist non zero polynomials such that:

$$a(x)b(x) = 1 \text{ mod}[p(x)]$$

At that time, a and b are multiplicative inverses and we see that a polynomial can be the inverse of another one. If the polynomial $p(x)$ is irreducible, the set of degree $n - 1$ polynomials along with the addition and multiplication modulo $p(x)$ forms a field.

Example: In $\mathbb{R}$, the polynomial $p(x) = x^2 + 1$ is irreducible. The set of polynomials of degree 1 with addition and multiplication modulo $p(x)$ forms a field. This field is called an extension of the field $\mathbb{R}$. Given two polynomials of degree one: $a(x) = a_0 + a_1x$ and $b(x) = b_0 + b_1x$, the product gives:

$$(a_0 + a_1x)(\, b_0 + b_1x) \text{ mod } (x^2 + 1) = (a_0b_0 - a_1b_1) + (a_0b_1 + a_1b_0)x$$

If we change the name of the indeterminate from x to j, we see that the above is the definition of the product of two complex numbers. In fact, from an algebraic point of view, the field of complex numbers $\mathbb{C}$ is just an extension of the field of real numbers, if we consider that a complex number is a pair of real coefficients of a first degree polynomial. At that time $\mathbb{R}[x]$ is the set of polynomials with real coefficients and $\mathbb{C}[x]$ is the set of polynomials with complex coefficients (which are themselves degree one polynomials). We can also say that the real numbers are degree zero polynomials and as such the field $\mathbb{R}$ is contained in $\mathbb{C}$.

Irreducible polynomials over GF(2)

In $\mathbb{R}$, the only irreducible polynomials are second degree polynomials with negative discriminant $\Delta < 0$. All other polynomials (degree ≥ 2) can be factored. This is not the case for polynomials over finite prime finite fields. In GF(q), q prime, we can find irreducible polynomials at all degrees larger than one. For example, over GF(2), the following polynomials are irreducible: $x^2 + x + 1$, $x^3 + x + 1$, $x^3 + x^2 + 1$, etc.

These polynomials do not have roots in GF(2), just as $x^2 + 1$ does not have roots in $\mathbb{R}$. However, it has roots in $\mathbb{C}$. If we consider the set of polynomials with real coefficients, a degree n polynomial has always n roots, not all real. The same theorem applies to polynomials with coefficients in some prime finite field. The roots of irreducible polynomials belong to an extension field.

Extension fields of GF(2)

The same way as for the set of complex numbers which has been derived as an extension of the set of real numbers, we can define extension fields using irreducible polynomials over GF(2). For example, $GF(2^2)$ is the field of first degree polynomials if the multiplication is taken modulo $x^2 + x + 1$. Its elements are 0, 1, x and $x + 1$ written as polynomials. We can also list them as a pair of coefficients (0, 0), (0, 1), (1, 0) and (1, 1). Here also, it is clear that the elements of GF(2) are degree 0 polynomials. When we want to list the elements of $GF(2^2)$ it is better to change the name of the indeterminate. Let us use α for example. The elements will be 0, 1, α and $\alpha + 1$.

In this field the addition table is:

+	**0**	**1**	α	$\alpha + 1$
0	0	1	α	$\alpha + 1$
1	1	0	$\alpha + 1$	α
α	α	$\alpha + 1$	0	1
$\alpha + 1$	$\alpha + 1$	α	1	0

And the multiplication table is:

.	**0**	**1**	α	$\alpha + 1$
0	0	0	0	0
1	0	1	α	$\alpha + 1$
α	0	α	$\alpha + 1$	1
$\alpha + 1$	0	$\alpha + 1$	1	α

Roots of polynomials

We have seen already the definition of a root. In GF(2), a degree n polynomial has always n roots, some of them are in extension fields. For example, $x^2 + x + 1$ is irreducible, it has no root in GF(2). However, if α is a root, it satisfies $\alpha^2 + \alpha + 1 = 0$. So, $\alpha^2 = \alpha + 1$. For this given polynomial, all the non-zero elements of the field can be generated by successive powers of the root. $\alpha^0 = 1$, $\alpha^1 = \alpha$, $\alpha^2 = \alpha + 1$. This means that α is a primitive element of the multiplicative group.

Primitive polynomial: An irreducible polynomial with a root that is primitive is called a primitive polynomial. The polynomial $x^2 + x + 1$ is primitive. If we generate an extension of GF(2) using a primitive polynomial, we can list all its non zero elements as powers of the primitive root (from zero to $2^n - 2$). So, the elements of GF(2^2) are: 0, $\alpha^0 = 1$, $\alpha^1 = \alpha$ and $\alpha^2 = \alpha + 1$. The multiplication table becomes very simple: $\alpha^i \alpha^j = \alpha^{(i+j) \bmod (q-1)}$ where $q = 2^n$ is the number of elements of the extension field. If a polynomial of degree m is primitive, its root is

of order $n = 2^m - 1$. So, its root is also a root of $x^n + 1$. This is also a characterization of primitive polynomials.

If we use an irreducible polynomial of a degree n, we can define an extension GF(2^n).

Example: Consider the field generated by the primitive polynomial: $p(x) = x^3 + x + 1$, GF(2^3).

Its root satisfies: $\alpha^3 = \alpha + 1$. So, the different powers of the root are:

power	polynomial	coefficients
0	0	0 0 0
α^0	1	0 0 1
α^1	α	0 1 0
α^2	α^2	1 0 0
α^3	$\alpha + 1$	0 1 1
α^4	$\alpha^2 + \alpha$	1 1 0
α^5	$\alpha^2 + \alpha + 1$	1 1 1
α^6	$\alpha^2 + 1$	1 0 1

We remark that $\alpha^7 = 1$. The first row of the above table is just the zero element of the field (it cannot be represented as a power of the root). The three columns represent the same entities. The first column is more useful for multiplication. The second (and third) are more useful for addition.

We have seen that the number of roots of a polynomial over GF(2) is equal to its degree. Another property of polynomials over GF(2) is the fact that if β is a root of a polynomial, β^2 is also a root.

Proof:

For polynomials over GF(2), we have $\left[a(x)\right]^2 = a\left(x^2\right)$.

$a(x) = a_0 + a_1 x + \cdots + a_n x^n$ and $c(x) = \left[a(x)\right]^2 = \left(a_0 + a_1 x + \cdots + a_n x^n\right)\left(a_0 + a_1 x + \cdots + a_n x^n\right)$.

The coefficients of the polynomial $c(x)$ can be computed by the convolution:

$$c_k = \sum_{i=0}^{k} a_i a_{k-i}$$

If k is odd, the number of terms in the above summation is even and we can write:

$c_k = \sum_{i=0}^{\frac{k-1}{2}} a_i a_{k-i} + \sum_{i=\frac{k+1}{2}}^{k} a_i a_{k-i}$. By making the change of index $l = k - i$ in the second term, we find that it is equal to the first one. So, the sum is zero and it means that the odd powers of the product are zero.

If *k* is even, the number of terms is odd and:

$c_k = \sum_{i=0}^{\frac{k}{2}-1} a_i a_{k-i} + a_{\frac{k}{2}} a_{\frac{k}{2}} + \sum_{i=\frac{k}{2}+1}^{k} a_i a_{k-i}$. We make the same change of index; we find that the first and the last term of the sum are identical. So, if *k* is even $c_k = a_{\frac{k}{2}} a_{\frac{k}{2}} = a_{\frac{k}{2}}$. So, the square of the polynomial *a*(*x*) is: $\left[a(x)\right]^2 = a_0 + a_1 x^2 + a_2 x^4 + \cdots + a_n x^{2n}$.

(Q.e.d)

Example:

Consider the polynomial $p(x) = x^3 + x + 1$. Its roots are all in GF(2^3) described in the previous table. They are: α , α^2 and $\alpha^4 = \alpha^2 + \alpha$. We can factorize it as $\left(x + \alpha\right)\left(x + \alpha^2\right)\left(x + \alpha^2 + \alpha\right)$.

If we consider the polynomial $p(x) = x^3 + x^2 + 1$, its roots are α^3 , $\alpha^6 = \alpha^2 + 1$ and $\alpha^{12} = \alpha^5 = \alpha^2 + \alpha + 1$.

In finite fields, there exist elements β such that $\beta^m = 1$. This element is said to be of order *m* and it is evident that it is a root of $x^m + 1$. If β is also a root of some polynomial *p*(*x*) of smaller degree, then *p*(*x*) must be a factor of $x^m + 1$.

Example:

If the previous field GF(2^3), we had $\alpha^7 = 1$. So, α is a root of $x^7 + 1$. This number is also a root of $x^3 + x + 1$. So $x^3 + x + 1$ is a factor of $x^7 + 1$. We have also $\left(\alpha^3\right)^7 = 1$, and α^3 is a root of $x^3 + x^2 + 1$. This polynomial is also a factor of $x^7 + 1$. The number 1 is also a root of $x^7 + 1$. Finally, we can factorize $x^7 + 1$ as: $x^7 + 1 = \left(x + 1\right)\left(x^3 + x + 1\right)\left(x^3 + x^2 + 1\right)$

Minimum polynomial

The minimum polynomial $m_\beta(x)$ is the least degree polynomial having β as a root.

Example:

Consider the previous field GF(2^3) generated by x^3+x+1 and the element α^2. If a polynomial has α^2 as a root, then it must have $\alpha^4=\alpha^2+\alpha$ and $\alpha^8=\alpha$ as roots. The next square is $\alpha^{16}=\alpha^9=\alpha^2$. So, the minimum polynomial of α^2 is:

$$m_{\alpha^2}(x)=\left(x+\alpha^2\right)\left(x+\alpha^4\right)\left(x+\alpha\right)=x^3+x+1$$

We can also show that $m_{\alpha^3}(x)=x^3+x^2+1$.

Introduction to coding

We have seen that Shannon's coding theorem predicts the existence of a code that allows quasi perfect transmission over a noisy channel. However, this theorem does not give any hint on how we can construct such code. In this chapter, we are going to see that we can improve communication by constructing efficient codes.

1. Basic definitions

An encoding procedure is the process of mapping sequences of symbols from a source alphabet to other sequences called codewords. In many cases, the basic alphabet of the information sequences and the one of the codewords is the same. In this chapter, we are going to limit ourselves to the binary alphabet. Since we are not in the information theory part of the course, we are going to call the binary digits "bits".
In the mathematical review, we have introduced the concept of finite fields. In this section, we are going to use either vectors defined over GF(2) or polynomials defined also over GF(2).

Terminology:

The information source is going to produce messages of length *k* bits and the encoder is going to add some redundancy to help combat noise. So, the codewords will have *n* bits, $n > k$. The ratio $R_c = \frac{k}{n}$ is called the "code rate". The code is designated as an (*n*, *k*) code.

In this first part, we are going to use linear algebraic techniques to show the process of coding.

2. Coding theory using vector space methods

In this part, the information sequence is a *k* dimensional vector over GF(2) (*k*-tuple).

$$\mathbf{i} = \begin{pmatrix} i_1 & i_2 & \cdots & i_k \end{pmatrix}$$

The codewords are *n* dimensional vectors over GF(2) (*n*-tuple).

$$\mathbf{c} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix}$$

The mapping between **i** and **c** can be linear or nonlinear. We are going to consider only linear codes because they are simpler to formalize. However, for any mapping, since $n > k$, we will

have always 2^k valid codewords chosen among 2^n possible n-tuples (if the mapping is injective). We can immediately see an error detection technique: If the received sequence is not a valid codeword, then some bits have changed during the process of transmission and we have errors. However, an exhaustive search over all possible bit combinations is very expensive in processing time. If we use linear mapping between the information sequences and the codewords, we can find some useful methods that will save processing time and complexity.

Hamming distance and Hamming weight

The Hamming weight $w(\mathbf{c})$ of an n dimensional vector $\mathbf{c}$ over GF(2) is the number of 1's in the vector. The Hamming distance $d(\mathbf{c}_i, \mathbf{c}_j)$ between two vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ is the number of places by which they differ. The Hamming weight can be used as a norm in the vector space of n-tuples. The distance and the weight are related as

$$d(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{c}_1 + \mathbf{c}_2) \tag{1}$$

Example: Let $\mathbf{c}_1 = (1 \quad 0 \quad 1 \quad 1)$ and $\mathbf{c}_2 = (1 \quad 1 \quad 0 \quad 1)$, we have $w(\mathbf{c}_1) = 3$, $w(\mathbf{c}_2) = 3$, $d(\mathbf{c}_1, \mathbf{c}_2) = 2$ and $\mathbf{c}_1 + \mathbf{c}_2 = (0 \quad 1 \quad 1 \quad 0)$ so $w(\mathbf{c}_1 + \mathbf{c}_2) = 2$.

The Hamming distance is an important factor in the error correction and detection capability of a code. It is left as an exercise to show that the Hamming distance satisfies all the requirements for a distance. Furthermore, as we have used the minimum Euclidian distance for the union bound, we can also define the minimum Hamming distance of a code.

$$d_{\min}^H = \min_{\substack{i,j \\ i \neq j}} d(\mathbf{c}_i, \mathbf{c}_j) \tag{2}$$

This distance determines the error correction and detection capability of a given code.

Consider two codewords $\mathbf{c}_1$ and $\mathbf{c}_2$ separated by a distance $d_{\min}^H$. We assume that $\mathbf{c}_1$ is transmitted. We receive an n-tuple $\mathbf{r}$ distant from $\mathbf{c}_1$ by t places (in the direction of $\mathbf{c}_2$). If $t = d_{\min}^H$, $\mathbf{r}$ might be confused with $\mathbf{c}_2$ and we cannot detect any error. So, we can say that this code is able to detect up to $d_{\min}^H - 1$ errors. In order to consider its error correction capability, we have to use an error correction algorithm. We will use a minimum distance classifier. That means that if $\mathbf{r}$ is closer to $\mathbf{c}_1$ than any other codeword, then we decide that it is $\mathbf{c}_1$ that has been transmitted. We have to consider two cases:

If $d_{\min}^H$ is even, then if $t = \frac{d_{\min}^H}{2}$, then the received word will be at equal distance from $\mathbf{c}_1$ and $\mathbf{c}_2$. So, in this case, we must have $t \le \frac{d_{\min}^H}{2} - 1$ in order to be able to correct the errors. If $d_{\min}^H$ is odd, then the received word situated at $t = \frac{d_{\min}^H - 1}{2}$ is closer to $\mathbf{c}_1$ and the one situated at $t = \frac{d_{\min}^H + 1}{2}$ will be closer to $\mathbf{c}_2$. So, for both cases, we can say that we can correct $t = \frac{\lfloor d_{\min}^H - 1 \rfloor}{2}$ errors[1].

Notion of coding gain

We want to transmit an information sequence $\mathbf{i} = (i_1 \quad i_2 \quad \cdots \quad i_k)$ of length *k*, i_j being a binary number. If we transmit directly this sequence using antipodal signaling, the energy per bit is E_{bu} and the bit duration is T_{bu}. Using signal space concepts, this amounts to transmit a point in a *k* dimensional space. So, to the sequence $\mathbf{i}_i = (i_{i1} \quad i_{i2} \quad \cdots \quad i_{ik})$ corresponds a point: $\mathbf{v}_i = \sqrt{E_{bu}}\,(v_{i1} \quad v_{i2} \quad \cdots \quad v_{ik})^T$ where $v_{ij} = \pm 1$ depending on $i_{ij} = 0 \text{ or } 1$. The duration of the transmitted symbol (the whole sequence) is $T = kT_{bu}$ and the energy corresponding to this sequence is $E = kE_{bu}$.

If we encode this bit sequence, we obtain a codeword $\mathbf{c}_i = (c_{i1} \quad c_{i2} \quad \cdots \quad c_{in})$ having *n* bits. If we use the same technique for transmission, the transmitted vector is now: $\mathbf{s}_i = \sqrt{E_{bc}}\,(\sigma_{i1} \quad \sigma_{i2} \quad \cdots \quad \sigma_{in})^T$ along with $\sigma_{ij} = \pm 1$ depending on $c_{ij} = 0 \text{ or } 1$. E_{bc} is the bit energy for the coded word and now the bit duration becomes $T_{bc} = T/n$ and the transmitted symbol energy is $E = nE_{bc}$.

We have 2^k codewords to be chosen among 2^n possible bit combinations. This corresponds to 2^k points located on vertices of an n-dimensional hypercube of edge length $2\sqrt{E_{bc}}$. The Hamming distance between two codewords $\mathbf{c}_i$ and $\mathbf{c}_j$ is d_{ij}^H. This means that

[1] $\lfloor \quad \rfloor$ stands for the floor function, i.e. the integer part of the argument.

there are d_{ij}^H bits that differ between the two codewords. The Euclidian distance between the corresponding vectors is given by:

$$\left(d_{ij}^E\right)^2 = \left\|\mathbf{s}_i - \mathbf{s}_j\right\|^2 = E_{bc}\left[\left(\sigma_{i1}-\sigma_{j1}\right)^2 + \left(\sigma_{i2}-\sigma_{j2}\right)^2 + \cdots + \left(\sigma_{in}-\sigma_{jn}\right)^2\right]$$

We have $\sigma_{il} - \sigma_{jl} = 0$ if the bits at the position *l* of two codewords $\mathbf{c}_i$ and $\mathbf{c}_j$ are the same and $\left|\sigma_{il} - \sigma_{jl}\right| = 2$ if they are different. So:

$$\left(d_{ij}^E\right)^2 = \left\|\mathbf{s}_i - \mathbf{s}_j\right\|^2 = E_{bc}\left(\underbrace{2^2 + 2^2 + \cdots + 2^2}_{d_{ij}^H \text{ terms}}\right)$$

Finally, $\left(d_{ij}^E\right)^2 = 4E_{bc}d_{ij}^H$ and the minimum distances are also related by $\left(d_{\min}^E\right)^2 = 4E_{bc}d_{\min}^H$. If we use the same energy to transmit a *k* bit uncoded message or an *n* bit coded one, then the following relation holds between the energy of uncoded bit and the one for the coded one:

$E = kE_{bu} = nE_{bc}$, so $E_{bc} = \frac{k}{n}E_{bu} = R_c E_{bu}$ where $R_c = \frac{k}{n}$ is the code rate. Using this result, we obtain: $\left(d_{\min}^E\right)^2 = 4R_c d_{\min}^H E_{bu}$.

We can now use the union bound for the probability of error:

$\Pr[E] \le \frac{M-1}{2}\operatorname{erfc}\left(\frac{d_{\min}^E}{2\sqrt{N_0}}\right)$ where $M = 2^k$ and we obtain finally:

$$\Pr[E] \le \frac{M-1}{2}\operatorname{erfc}\left(\sqrt{\frac{R_c d_{\min}^H E_{bu}}{N_0}}\right)$$

If we do not use coding, then the probability of error is bounded by:

$$\Pr[E] \le \frac{M-1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_{bu}}{N_0}}\right)$$

So, we see that we can define a "coding gain" $G_c = R_c d_{\min}^H$ (3)

Example:

If we use a (3, 1) repetition code, we have $R_c = \frac{1}{3}$ and $d_{\min}^H = 3$, this gives $G_c = 1$, no gain!

However, for a (7, 4) Hamming code[2], $R_c = \frac{4}{7}$ and $d_{\min}^H = 3$, $G_c = \frac{12}{7} > 1$.

[2] These codes will be defined later.

This result means also that we can tolerate quite slow codes (R_c small) if $d_{\min}^{H}$ is large.

Linear codes

A code is said to be linear if the mapping from the information space (k dimensional) to the code space (n dimensional) is linear. Since these two spaces are finite, then we can represent this mapping by a matrix. In coding theory, it is customary to use row vectors to represent n-tuples. So, we define a "generator" matrix **G** having k rows and n columns.

We have: $\mathbf{c} \in [\mathrm{GF}(2)]^n$, $\mathbf{i} \in [\mathrm{GF}(2)]^k$ so

$$\mathbf{c} = \mathbf{iG} \tag{4}$$

The matrix **G** has k rows and n columns and if the mapping is injective (we have 2^k distinct codewords), the matrix has a rank equal to k. Its rows constitute a basis in the k dimensional subspace containing the codewords (the code space).

Example:

(3, 1) repetition code: In this coding method, we repeat the bit three times. So, $\mathbf{i} \in \{0,1\}$ and $\mathbf{c} \in \{000,111\}$. The matrix **G** is the following row vector: $\mathbf{G} = (1 \quad 1 \quad 1)$. The error correction proceeds by using the majority rule to decide on the transmitted bit.

(n, n-1) single bit parity check code: In this coding method, we append a parity check bit making the overall number of ones in the codeword even. It is evident that this method can detect errors but it cannot correct any one. The generator matrix is

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

A (3, 2) code generator matrix is: $\mathbf{G} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$, we have 4 codewords:

i	**c**
00	000
01	011
10	101
11	110

We can remark that the vector **i** is repeated in **c** and a single parity check bit is appended, making the overall parity (number of ones) even.

(7, 4) Hamming code:

The generator matrix is: $\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$

In this code, the information sequence forms the first four bits of the codeword and the last three bits are computed from the information bits. We see that the codeword is:

$\mathbf{c} = \begin{pmatrix} i_1 & i_2 & i_3 & i_4 & i_1+i_3+i_4 & i_1+i_2+i_3 & i_2+i_3+i_4 \end{pmatrix}$ for $\mathbf{i} = \begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix}$

Properties of linear codes

The first important property of a linear code is that the sum of any two codewords is also a codeword. Let $\mathbf{c}_1 = \mathbf{i}_1\mathbf{G}$ and $\mathbf{c}_2 = \mathbf{i}_2\mathbf{G}$, so, $\mathbf{c}_1 + \mathbf{c}_2 = (\mathbf{i}_1 + \mathbf{i}_2)\mathbf{G}$. However, the sum of two information vectors is always some information vector. So, $\mathbf{i}_1 + \mathbf{i}_2 = \mathbf{i}_m$, $\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{i}_m\mathbf{G} = \mathbf{c}_m$ and $\mathbf{c}_m$ is a codeword by definition.

Since we require that the mapping defining the code be injective, then the rows of **G** are linearly independent. They can be used as a basis for the *k* dimensional space of the codewords. If we express the matrix **G** as:

$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{pmatrix}$; $\mathbf{g}_m$ being a row of **G** and $\mathbf{i} = \begin{pmatrix} i_1 & i_2 & \cdots & i_k \end{pmatrix}$

then
$$\mathbf{c} = \sum_{m=1}^{k} i_m \mathbf{g}_m \tag{5}$$

One important property of linear codes is the fact that the minimum Hamming distance is equal to the minimum Hamming weight of the non-zero codewords.

By definition, $d_{\min}^{H} = \min_{\substack{i,j \\ i \neq j}} d(\mathbf{c}_i, \mathbf{c}_j) = \min_{\substack{i,j \\ i \neq j}} w(\mathbf{c}_i + \mathbf{c}_j) = \min_{\substack{m \\ \mathbf{c}_m \neq 0}} w(\mathbf{c}_m)$

For the three examples seen above, we have:

The (3, 1) repetition code has $d_{\min}^{H} = 3$. It can correct single errors and detect double errors. However, we have seen that it does not improve the overall communication system.

The (*n*, *n*–1) single bit parity check code has $d_{\min}^{H} = 2$. It cannot correct errors but it can detect single errors.

The (7, 4) Hamming code has $d_{\min}^{H} = 3$. It can correct single errors and detect double errors just as the (3, 1) repetition code. However, we have seen that this code presents a coding gain.

Another important property of the linear codes is the fact that elementary row operations on the matrix **G** (interchange of two rows, addition of two rows) will produce the same subspace and we will have the same codewords. This can be seen from equation(5). Exchange of columns on the other hand will produce equivalent codewords (the distance properties do not change). In this case, it is the order of the bits that will change.

Systematic codes

In many cases, the codeword consists of the information sequence to which some parity check bits are appended. In other words and using $r = n - k$:

$$\mathbf{c} = \begin{pmatrix} i_1 & i_2 & \cdots & i_k & p_1 & p_2 & \cdots & p_r \end{pmatrix} \tag{6}$$

This implies that the generator matrix has the following shape:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1r} \\ 0 & 1 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2r} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k1} & p_{k2} & \cdots & p_{kr} \end{pmatrix}$$

Using block matrices, we can express it as:

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{P} \end{pmatrix} \tag{7}$$

where **I** is a $k \times k$ identity matrix and **P** is $k \times r$ matrix.

We can always transform a non-systematic code to a systematic one by performing elementary row operations and column permutations (column exchange). We have seen that the row operations will produce the same codewords (not in the same order), while the column exchange will produce equivalent codes (same Hamming weight). These transformations are left multiplication of the matrix **G** by some $k \times k$ elementary matrix for row operations or right multiplication by an $n \times n$ permutation matrix for row exchange.

Row switching (permutation) elementary matrix:

It consists of an identity matrix with row *i* and row *j* swapped.

$$\mathbf{S}_{ij} = \begin{pmatrix} 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & & & \cdots & & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ row\ i \\ \\ \\ \updownarrow \\ \\ row\ j \\ \\ \\ \\ \end{matrix}$$

Row adding elementary matrix (addition of row *i* and row *j*):

It consists of an identity matrix with a 1 added at the position (*i*, *j*).

$$\mathbf{A}_{ij} = \begin{pmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & & & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ row\ i \\ \\ \\ \end{matrix}$$

$$\begin{matrix} column\ j \\ \downarrow \end{matrix}$$

Of course, these matrices will act on rows if they are multiplied from the left and on the columns if they are multiplied on the right.

We can also remark that the inverse of the permutation matrix $\mathbf{S}_{ij}$ is the matrix itself. Its use a second time will simply undo the row (column) exchange.

Example:

Consider the following generator matrix:

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

If we exchange the first and fourth column, we obtain:

$$\mathbf{GS}_{14} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

We now swap column 3 and column 4:

$$\mathbf{G}' = \mathbf{GS}_{14}\mathbf{S}_{34} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The matrices $\mathbf{S}_{ij}$ are 7×7 matrices. The new generator matrix is systematic and the codewords generated by **G**' are related to the ones generated by **G** by: $\mathbf{c}' = \mathbf{cS}_{14}\mathbf{S}_{34}$ and we have $\mathbf{c} = \mathbf{c}'\left(\mathbf{S}_{14}\mathbf{S}_{34}\right)^{-1} = \mathbf{c}'\mathbf{S}_{34}\mathbf{S}_{14}$. In other words, to recover **c** from **c**', we must swap the bit positions in the reverse order {(3, 4) first and then (1, 4)}.

Parity Check Matrix

We have seen that the code subspace is a *k* dimensional space spanned by the rows of the generator matrix. Since the codewords are *n* dimensional vectors, there exists an $n - k = r$ dimensional space that is orthogonal to it (dual space). All vectors of this dual space are orthogonal to the codewords and since its dimension is *r*, we can define an *r*×*n* matrix **H** such that for any codeword, we can write:

$$\mathbf{cH}^T = \mathbf{0} \tag{8}$$

The rows of **H** are orthogonal to all codewords and in particular, they are orthogonal to the rows of **G**. We have:

$$\mathbf{GH}^T = \mathbf{0} \tag{9}$$

Equation (8) is called a parity check equation and the matrix **H** is called the parity check matrix. We can use equation (8) to verify that a received n tuple is a codeword. If the code is systematic, it is easy to derive the parity check matrix from the generator one. We have seen that the generator matrix is given by equation(7): $\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{P} \end{pmatrix}$. In order to satisfy equation(9), the matrix **H** must be:

$$\mathbf{H} = \begin{pmatrix} \mathbf{P}^T & \mathbf{I} \end{pmatrix} \tag{10}$$

where **P** is an *r*×*k* matrix and **I** is an *r*×*r* identity matrix. So, one technique for deriving the parity check matrix from **G** is to transform **G** to systematic form, derive the associated parity

check matrix using (10) and apply to it the inverse transformation (in the case of column swap).

Example:

Let consider the previous generator matrix: $\mathbf{G} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$.

The associated systematic matrix is: $\mathbf{G}' = \mathbf{G}\mathbf{S}_{14}\mathbf{S}_{34} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$.

Using(10), we obtain the following parity check matrix:

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

$\mathbf{H}'$ is the parity check matrix associated with $\mathbf{G}'$. We can write $\mathbf{G}'\mathbf{H}'^{T} = \mathbf{0}$. So, we obtain: $\mathbf{G}\mathbf{S}_{14}\mathbf{S}_{34}\mathbf{H}'^{T} = \mathbf{0}$. So, $\mathbf{H}^{T} = \mathbf{S}_{14}\mathbf{S}_{34}\mathbf{H}'^{T}$ or $\mathbf{H} = \mathbf{H}'\mathbf{S}_{34}^{T}\mathbf{S}_{14}^{T} = \mathbf{H}'\mathbf{S}_{34}\mathbf{S}_{14}$. The last equality comes from the fact that the swap matrices are symmetrical. It also means that we recover the parity check matrix from $\mathbf{H}'$ by swapping first the third and fourth column and then the first and fourth column of the result. The parity check matrix associated with $\mathbf{G}$ is:

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

Row operations on the other hand do not change the subspace spanned by the rows of $\mathbf{G}$. So, any vector that is orthogonal to a codeword remains orthogonal after transformation of these codewords. Let $\mathbf{G}' = \mathbf{B}\mathbf{G}$. Then a matrix $\mathbf{H}$ such $\mathbf{G}'\mathbf{H}^{T} = \mathbf{0}$ satisfies also $\mathbf{G}\mathbf{H}^{T} = \mathbf{0}$ when the matrix $\mathbf{B}$ represents an elementary row operation (it can be inverted).

Example:

Consider the matrix $\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$. It can be transformed to systematic form by interchanging the second and the third row. So, we obtain:

$$\mathbf{G}' = \mathbf{BG} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The matrix **H** is then:

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

The fact that $\mathbf{cH}^T = \mathbf{0}$ implies that we can use the matrix **H** to find the minimum distance of the code.

Theorem:

The minimum Hamming distance $d_{\min}^H$ of the code is equal to minimum number of columns of the parity check matrix **H** that add to zero (that are linearly dependent).

Proof:

We have seen that the minimum distance is equal to the minimum Hamming weight of the non zero codewords. If we consider a codeword with a weight $w(\mathbf{c}) = d_{\min}^H$, it has exactly $d_{\min}^H$ ones at the positions $i_1, i_2, \ldots, i_{d_{\min}^H}$. Let us express the parity check matrix as: $\mathbf{H} = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n \end{pmatrix}$ where $\mathbf{h}_i$ are vectors equal to the columns of **H**. $\mathbf{cH}^T = \mathbf{0}$ translates to: $\sum_{k=1}^{d_{\min}^H} \mathbf{h}_{ik}^T = \mathbf{0}$. It is clear that we cannot have a sum having less than $d_{\min}^H$ terms as it would mean that there exist a codeword with a Hamming weight less than $d_{\min}^H$.

(Q.e.d.)

The fact that **H** is of rank $r = n - k$ implies the following inequality (Singleton Bound):

$$d_{\min}^H \leq r + 1 \tag{11}$$

Proof:

The rank of **H** is *r*. The row rank is equal to the column rank. This means that some combination of $r + 1$ columns of **H** are linearly dependent. The previous theorem states that $d_{\min}^H$ is the minimum number of linearly dependent columns of **H**.

(Q.e.d.)

A code satisfying the above relation (11) with equality is called "maximum distance separable" code.

Example:

Consider the (7, 4) Hamming code. Its parity check matrix is:

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

It is clear that we cannot find any two different columns that add to zero. However, several combinations of 3 columns add to zero. For example, the first, second and fourth column add to zero.

The previous (7, 3) code has

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

This code has $d_{\min}^{H} = 4$ and this can be seen from its parity check matrix. The first, second, fifth and sixth columns add to zero. We can verify it by listing all the codewords:

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

All non zero codewords have a Hamming weight of four.

Dual codes

If we have a code with a generator matrix **G** and a parity check matrix **H**, there exist a dual code with a generator matrix **H** and a parity check matrix **G**.

Example:

The (3, 1) repetition code has:

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \text{ and } \mathbf{H} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Its dual code is the single bit parity check code (3, 2).

Syndrome[3]

When a codeword is transmitted, there will be noise and interference in the transmission that will change some bits (0 becomes 1 and vice versa). In many transmission systems, we can model this by a BSC channel with error probability *p*. So, the received vector will be different from the transmitted one and we can write:

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

In the above equation **r** is the received vector, **c** is the transmitted one and **e** is the error pattern (vector). Of course, all vectors are n tuples. The error pattern contains a 1 at the position of an error and a zero elsewhere (in GF(2), when we add 1, we invert the value). The example below shows a single error at the i^{th} position.

$$\mathbf{e} = \begin{pmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{pmatrix}$$
$$\uparrow$$
$$i^{th}\ position$$

If we apply equation (8) to the received vector, we obtain an *r* dimensional vector **s** called "syndrome".

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T \tag{12}$$

Clearly, if there is no error in the transmission, the result should be a zero vector. In fact, it is easy to show that the syndrome does not depend on the transmitted codeword, but only on the error pattern that has occurred during that transmission.

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T = (\mathbf{c} + \mathbf{e})\mathbf{H}^T = \mathbf{e}\mathbf{H}^T \tag{13}$$

So, the syndrome gives us some information about the errors that have occurred during the transmission. However, this information is incomplete. The dimension of the syndrome vector is $r = n - k$ and the dimension of the error pattern is *n*. From this, we see that we cannot correct all errors that will occur in the transmission but only the most probable ones. In fact, from(13), we see that there are 2^k error patterns that have the same syndrome.

Error detection

If the syndrome is different from zero, this means that there were some errors during the transmission. However, there can be some errors that cannot be detected. If the error vector is

[3] A syndrome in medicine is an ensemble of symptoms characterizing a disease.

equal to a codeword different from zero, we have an undetectable error. The probability of such event can be computed. We must have the weight distribution of the code.

Definition:

Let A_i be the number of codewords with a weight i. The numbers $A_0, A_1, \ldots, A_n$ are called the weight distribution of the code.

Example: Consider the (7, 4) Hamming code.

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

The 16 codewords are listed above. We have:

$$A_0 = 1; A_1 = A_2 = 0; A_3 = 7; A_4 = 7; A_5 = A_6 = 0; A_7 = 1$$

If we use a BSC for transmission with a probability of error p, we can compute the probability that an error vector is equal to a non-zero codeword:

$$\Pr\left[\textit{undetectable error}\right] = \Pr\left[\mathbf{e} = \mathbf{c}\right] = \sum_{i=d_{\min}^{H}}^{n} A_i p^i (1-p)^{n-i}$$

In the previous case, we have $\Pr = 7 \times p^3(1-p)^4 + 7 \times p^4(1-p)^3 + p^7$.

For $p = 10^{-3}$, we obtain: $\Pr \approx 7 \times 10^{-9}$

The syndrome is also related to the parity check matrix as follows:

Let the error vector be $\mathbf{e} = \begin{pmatrix} e_1 & e_2 & \cdots & e_n \end{pmatrix}$ and the parity check matrix be $\mathbf{H} = \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n \end{pmatrix}$, we have:

$$\mathbf{s} = \mathbf{e}\mathbf{H}^T = \sum_{i=1}^{n} e_i \mathbf{h}_i^T \tag{14}$$

This means that the syndrome transposed is the sum of the columns of **H** where there is an error. So, if there is a single error, the syndrome transposed is equal to the column indexed by the location of the error. This property can be used to design single error correcting codes.

Hamming codes

These codes are single error correcting codes. They have a minimum distance $d_{\min}^{H} = 3$. They are characterized by having columns of **H** equal to all non-zero (maximum number of) binary numbers of size $r = n - k$. It means that the **H** matrix is an $r \times \left(2^r - 1\right)$ matrix and that $k = 2^r - 1 - r$. So, the Hamming codes are $\left(2^r - 1, 2^r - 1 - r\right)$ codes.

When r = 2, the code is a (3, 1) repetition code.

When r = 3, the code is the (7, 4) code that we have already seen.

When r = 4, the code is a (15, 11) code with a parity check matrix **H** given by:

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

Hamming sphere and Hamming bound

Given a codeword **c**, the number of n tuples that are at a distance d from **c** is $\binom{n}{d}$.

The Hamming sphere of radius t and center **c** is the set of all the n tuples that are situated at a distance less or equal to than t from **c**. The number of such vectors (volume of the sphere) is given by:

$$V\left(n,t\right) = \sum_{k=0}^{t} \binom{n}{k} \tag{15}$$

In the above volume, we are counting also **c** itself (distance zero).

The Hamming bound is a relation between the total number of syndromes and the above volume. It is evident that we must have more syndromes than correctable errors. The total number of syndrome is 2^r. So, if the code can correct t errors, we must have:

$$2^r \geq \sum_{k=0}^{t} \binom{n}{k} \tag{16}$$

If we have equality in (16), the code is called "perfect".

The only binary perfect codes are: odd length repetition codes, Hamming codes and the Golay (23, 12) code[4].

In perfect codes, we have exactly the same number of syndromes as correctable error patterns. In most error correcting codes, we have more syndromes than correctable error patterns.

Error correction

There are essentially two strategies for error decoding. We can use "soft decoding". In this case, the 2^k codewords are mapped to 2^k different waveforms. We can use MAP receiver that will detect optimally the correct waveform. It is evident that this technique is optimal. However, it is usually very costly. The other way is "hard decoding". This is a sub optimal technique. It consists of demodulating the received signal into a sequence of ones and zeros. We have then to find whether some of the bits have changed during the transmission or not. If they have changed, we must correct the assumed error.

In this part, we are going to study only the hard decoding.

Maximum likelihood decoding

The first technique that comes to mind is to use a minimum distance classifier. It we receive a vector $\mathbf{r}$, we select the closest codeword $\mathbf{c}_i$ as being the correct one.

If we use this method, we have to test all the 2^k possible codewords and this can be prohibitive. If $k = 200$, we will have to test 2^{200} codewords and this is clearly impossible.

If we use a BSC with probability p and a t error correcting (n, k) code, the probability that the error pattern contains more than t ones is:

$$\sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

The real probability of error is smaller or equal.

Syndrome table decoding

We have seen that the syndrome depends only on the error pattern. One technique for decoding is to build a table containing the most probable error patterns (having up to t ones) addressed by their corresponding syndrome. So, the error correcting will proceed by first computing the syndrome. Then, derive the corresponding error pattern with the smallest number of ones and finally XOR this pattern with the received word.

---

[4] We will return to that code when we will see the cyclic codes.

Example:

Consider the following (6, 3) code with the generator matrix:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \text{ and parity check matrix } \mathbf{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This code has $d_{\min}^{H} = 3$. So, it is a single error correcting code. Since $r = 3$, the Hamming bound is $2^3 = 8 > \binom{6}{0} + \binom{6}{1} = 7$. The code is not perfect. We have 6 single errors and 7 syndromes that are different from zero. We can derive the following table:

Syndrome	*error pattern*
000	000000
001	000001
010	000010
100	000100
110	001000
101	010000
011	100000
111	100100

We must not forget that the above error patterns are the most probable ones. We obtain the same syndromes if we add a codeword to the error pattern. The last row corresponds to two errors. In general it should not be used for error correction but for error detection.

The above example has a value of *r* that is quite small. The syndrome table is reasonable. However, we should resort to more complex techniques of error correction if we use high value of *n* and *k*.

3. Cyclic codes

The use of linear cyclic codes allows the realization of powerful error correcting and detecting codes. The hardware required for the implementation of these codes is much simpler than the one required for the previous codes.

Definition:

A cyclic code is a code that is invariant to cyclic shifts. It means that a cyclic shift applied to a codeword produces a codeword.

Cyclic shift:

Given the codeword $\mathbf{c} = (c_{n-1} \quad c_{n-2} \quad \cdots \quad c_1 \quad c_0)$, a cyclic shift by one position of **c** produces $\mathbf{c}' = (c_{n-2} \quad c_{n-3} \quad \cdots \quad c_0 \quad c_{n-1})$. In this part of the course, we change the indexing of the components of the vectors to comply with the polynomial notation that we are going to use from now on.

So, a linear cyclic code has two important properties: A cyclic shift of a codeword is a codeword and a sum of two codewords is also a codeword. The following list shows a (7, 4) cyclic code generated by the codeword $(0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1)$. The first codeword is of course the all zero vector (the code is linear), the other codes are all either cyclic shifts of previous codewords or sum of some codewords.

0	0	0	0	0	0	0	*zero codeword*
0	0	0	1	0	1	1	*generator*
0	0	1	0	1	1	0	*cyclic shift*
0	1	0	1	1	0	0	"
1	0	1	1	0	0	0	"
0	1	1	0	0	0	1	"
1	1	0	0	0	1	0	"
1	0	0	0	1	0	1	"
0	0	1	1	1	0	1	*sum of second and third*
0	1	1	1	0	1	0	*cyclic shift*
1	1	1	0	1	0	0	"
1	1	0	1	0	0	1	"
1	0	1	0	0	1	1	"
0	1	0	0	1	1	1	"
1	0	0	1	1	1	0	"
1	1	1	1	1	1	1	*sum of sixth and fifteenth*

The above code is in fact the (7, 4) Hamming code.

We are going to use polynomials to describe the different vectors involved in the coding procedure because the operation of shift can be described very simply using polynomials.

So, an (*n*, *k*) code entails the following polynomials over GF(2):

A degree $k-1$ information polynomial $i(x) = i_0 + i_1 x + \cdots + i_{k-2} x^{k-2} + i_{k-1} x^{k-1}$

A degree $n-1$ codeword polynomial $c(x) = c_0 + c_1 x + \cdots + c_{n-2} x^{n-2} + c_{n-1} x^{n-1}$

A cyclically shifted codeword is $c^{(1)}(x) = c_{n-1} + c_0 x + \cdots + c_{n-2} x^{n-1}$. If we compare it with $xc(x)$ (degree n), we obtain: $c^{(1)}(x) = xc(x) + c_{n-1} x^n + c_{n-1} = xc(x) + c_{n-1}(x^n + 1)$. So, if we apply the result of the Euclidian division, we can write:

$$c^{(1)}(x) = xc(x) \bmod \left[x^n + 1 \right] \tag{17}$$

We can generalize the above result to any number of cyclic shifts:

For j cyclic shifts, we have $c^{(j)}(x) = x^j c(x) \bmod \left[x^n + 1 \right]$

Generator polynomial

In the previous example, the "generator" codeword is $(0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1)$. In polynomial form, we can express it as: $g(x) = x^3 + x + 1$. It is left as an exercise to show that the codewords $c(x) = i(x)g(x)$ are the same as the ones of the above list, but in a different order.

For cyclic codes, we can generate the codewords using a polynomial $g(x)$ of degree r called the "generator" polynomial using polynomial multiplication. So, given information sequences represented by degree $k - 1$ polynomials $i(x)$, the codewords are $n - 1$ degree polynomials $c(x)$ computed using:

$$c(x) = i(x)g(x) \tag{18}$$

The generator polynomial $g(x) = g_r x^r + g_{r-1} x^{r-1} + \cdots + g_1 x + g_0$ must satisfy a certain number of conditions to qualify as a generator polynomial for a cyclic code.

Properties of a generator polynomial

1) The polynomial $g(x)$ must be a degree r polynomial. It means that $g_r = 1$.

2) The polynomial $g(x)$ must be a factor of $x^n + 1$.

Proof:

Consider a codeword produced by (18): $c(x) = i(x)g(x)$. A cyclic shift of $c(x)$ must also be a codeword and it is given by: $c'(x) = xc(x) \bmod \left[x^n + 1 \right] = i'(x)g(x)$ for some other information sequence $i'(x)$. So, $c'(x)$ is the remainder of the division of $xc(x)$ by $x^n + 1$. We can then write:

$$xc(x) = q(x)(x^n + 1) + c'(x) \tag{19}$$

The quotient $q(x)$ is a degree zero polynomial. Its value is either 0 when we do a simple shift ($c_{n-1} = 0$) or 1 when we perform a cyclic shift ($c_{n-1} = 1$). The equation (19) can be re-expressed as: $xc(x) + c'(x) = q(x)(x^n + 1) = \left[xi(x) + i'(x) \right] g(x) = q(x)(x^n + 1)$.

If $q(x) = 0$ then $i'(x)$ is a simple shift of $i(x)$. If $q(x) = 1$, we can write:

$$\left[xi(x)+i'(x)\right]g(x)=x^n+1$$

So, g(x) must be a factor of $x^n + 1$.

(Q.e.d)

3) The coefficient g_0 must be equal to one. Because it is a factor of $x^n + 1$, $g(x)$ cannot have zero as a root. So, $g(x)=x^r+\left(\sum_{i=1}^{r-1} g_i x^i\right)+1$.

Systematic cyclic codes

If we use the equation (18) to produce codewords, the code is clearly nonsystematic. One way to make sure that the code is systematic is to have the codeword polynomial be a sum of two polynomials: $c(x)=p(x)+x^r i(x)$. The polynomial $p(x)$ is a degree $r-1$ polynomial. So, the codeword polynomial can be written as:

$$c(x)=p_0+p_1x+\cdots+p_{r-1}x^{r-1}+i_0x^r+i_1x^{r+1}+\cdots+i_kx^{n-1}$$

One way to make certain that the polynomial $p(x)$ is of degree $r-1$ is to compute it as the remainder of some polynomial modulo $g(x)$, since $g(x)$ is a degree r polynomial. The systematic codeword is:

$$c(x)=x^r i(x)+x^r i(x)\bmod\left[g(x)\right] \tag{20}$$

Proof:

We just have to show that $c(x) = q(x)g(x)$ for some degree $k-1$ polynomial $q(x)$. The polynomial $p(x)=x^r i(x)\bmod\left[g(x)\right]$ is the remainder of the division of $x^r i(x)$ by $g(x)$ and is a degree $r-1$ polynomial. So, we can write: $x^r i(x)=q(x)g(x)+x^r i(x)\bmod\left[g(x)\right]$ where the quotient $q(x)$ is a degree $k-1$ polynomial or $x^r i(x)\bmod\left[g(x)\right]=x^r i(x)+q(x)g(x)$. Replacing in (20), we obtain: $c(x)=x^r i(x)+x^r i(x)+q(x)g(x)=q(x)g(x)$.

(Q.e.d)

The (23, 12) Golay code

The code is the only perfect three error corrector code. It has been used in many communication systems. The NASA mission Voyager has used the code. It can be generated by either $g_1(x)=x^{11}+x^9+x^7+x^6+x^5+x+1$ or $g_2(x)=x^{11}+x^{10}+x^6+x^5+x^4+x^2+1$. This is due to the fact that $x^{23}+1=(x+1)g_1(x)g_2(x)$.

Parity polynomial

We have seen that $g(x)$ is a factor of $x^n + 1$. So, we can write:

$$x^n + 1 = g(x)h(x) \tag{21}$$

where $h(x)$ is a degree k polynomial.

So, if we multiply a codeword $c(x)$ by $h(x)$, we obtain:

$$c(x)h(x) = i(x)g(x)h(x) = 0 \bmod \left[x^n + 1 \right] \tag{22}$$

This means that we can use the polynomial h(x) to check whether an $n - 1$ degree polynomial is a valid codeword or no. The polynomial $h(x)$ is called the "parity polynomial". This polynomial is also used in the generation of codewords. If we consider the equation(22), we can see that

$$\begin{aligned} c(x)h(x) = i(x)g(x)h(x) &= i(x)\left(x^n + 1\right) \\ &= i(x) + x^n i(x) \end{aligned}$$

In the above equation, the polynomial $i(x)$ is a degree $k - 1$ polynomial and $x^n i(x)$ starts at the coefficient of degree n. So, all the coefficients corresponding to degrees k up to $n - 1$ in $c(x)h(x)$ must be zero. So, we can write:

$$\sum_{i=0}^{k} h_i c_{n-i-j} = 0 \quad ; \; 1 \le j \le n-k \tag{23}$$

We have also seen in the first part of this chapter that for every code, there exists a dual code. The same thing applies to cyclic codes.

We have $h(x) = h_0 + h_1 x + \cdots + h_k x^k$. The polynomial $x^k h(x^{-1}) = h_0 x^k + h_1 x^{k-1} + \cdots + h_{k-1} x + h_k$ is also a factor of $x^n + 1$ and it generates an $(n, n - k)$ code such that all codeword are orthogonal to the codewords generated by $g(x)$. This code is the dual code of the (n, k) code generated by $g(x)$.

Hardware implementation of cyclic codes encoders

We have seen that the implementation of codes using the **G** matrix is achieved by a combinatorial structure and it can be very expensive (it consists of a large array of XOR gates). The implementation of cyclic codes can be achieved by the use of a sequential circuit. It is built around a shift register. The data appear sequentially.

Non Systematic Encoder

It consists of a straightforward implementation of equation(18). The coefficients of the product of $i(x)$ with $g(x)$ (coefficients of $c(x)$) are computed using the convolution equation:

$$c_m = \sum_{j=0}^{r} g_j i_{m-j} \tag{24}$$

So, we see that we just need to multiply shifted values of the input by the coefficients of the generator polynomial. In Figure 2, the blocks with the letter D represent D flip-flops, the adders are modulo 2 (XOR) adders and the branches labelled g_m indicate a multiplication by the generator polynomial coefficient g_m. In other words, if $g_m = 1$, the branch is implemented, otherwise, it is omitted. At start up, the registers are cleared. The information sequence is then shifted in least significant bit (LSB) first (i_0) for its k bits and r zeroes are appended. The codeword appears at the output LSB first for n clock pulses.

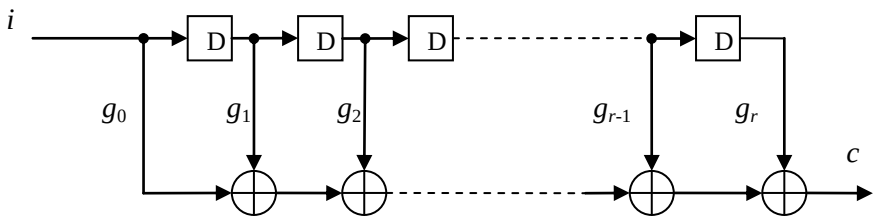


Figure 2 Non systematic encoder

Systematic encoders

Systematic cyclic codes use the encoding procedure exposed by equation(20). We first output the information sequence i and then the parity bits but this time, the data appears most significant bit (MSB) first. We need now a circuit that implements a division.

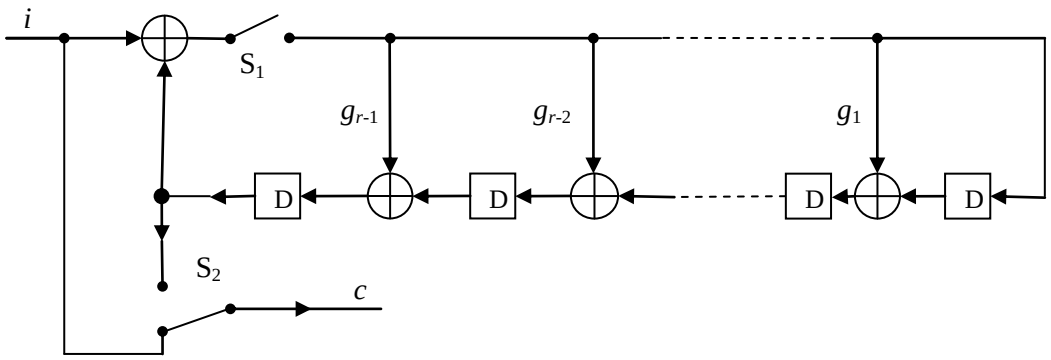


Figure 3 Systematic encoder using the generator polynomial

The encoding starts with the switch S_1 closed and S_2 in the shown position. The k information bits are shifted in the encoder MSB first and they appear at the output. After k clock pulses, the data bits are all shifted out, S_1 opens and S_2 is toggled in the other position. We need r clock pulses to shift out the content of the register which is the remainder of the division of $x^r i(x)$ by $g(x)$.

Example:

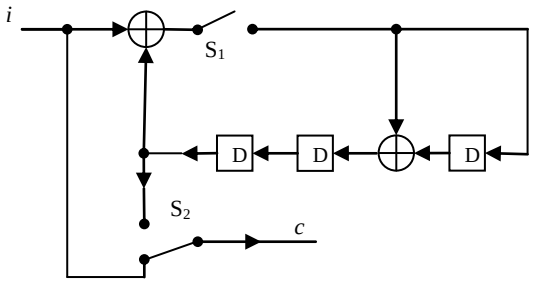


Figure 4 (7, 4) Hamming encoder

Figure 4 represents the systematic encoder for a cyclic (7, 4) Hamming encoder. The generator polynomial is $g(x) = x^3 + x + 1$. We want to transmit the four bit word (1 1 0 0) corresponding to the polynomial $i(x) = x^3 + x^2$. We have $r = 3$, so $x^3 i(x) = x^6 + x^5$ and $x^r i(x) \bmod [g(x)] = x$ corresponding to the three bit word (0 1 0). It is left as an exercise to show that with the switch S_1 closed, after shifting in the four bit word (1 1 0 0), the three flip-flops will contain the word (0 1 0) corresponding to the polynomial *x*. So, the codeword is (1 1 0 0 0 1 0).

Another structure for the encoder can be derived from equation(23) using the parity polynomial *h*(*x*). It is a difference equation and since the polynomial *h*(*x*) is of degree *k*, we must have $h_k = 1$. We can re-express (23) as:

$$c_{r-j} = \sum_{i=0}^{k-1} h_i c_{n-i-j} \qquad \text{for } 1 \le j \le r \tag{25}$$

For a systematic code, the bits c_r, c_{r+1}, …, c_{n-1} are the data bits ($i(x)$). The parity check bits can be computed using(25).

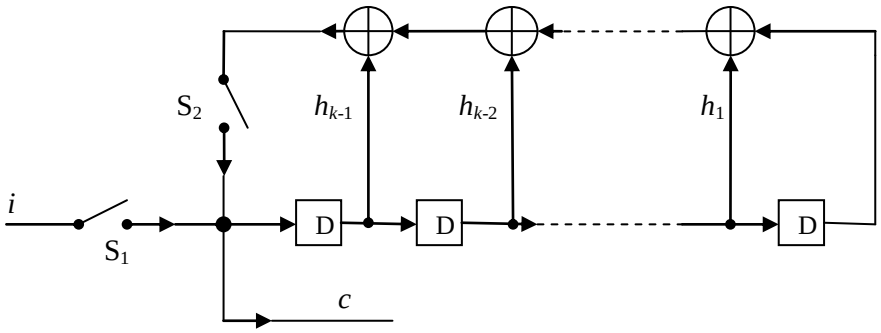
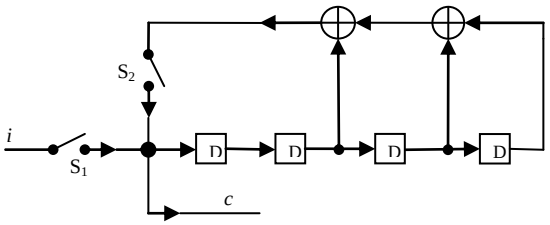


Figure 5 Systematic encoder using the parity polynomial

The encoding starts with the switch S_1 closed and S_2 open. After *k* clock pulses, the information bits are output and at the same time, shifted in the *k* flip-flops. After, S_1 opens and S_2 is closed. The parity check digits ($x^r i(x) \bmod [g(x)]$) are output.

Example: From the factorization $x^7+1=(x+1)(x^3+x^2+1)(x^3+x+1)$, we obtain the following parity polynomial for the (7, 4) Hamming code: $h(x)=x^4+x^2+x+1$. We obtain the following structure:



Generator and Parity Check Matrices for Cyclic Codes

The considered cyclic codes are linear codes and as such can be generated and checked using the methods seen in the previous part. That means that we can use the generator and parity check matrices.

Systematic code

Consider an (*n*, *k*) cyclic code with generator polynomial *g*(*x*). The generator is a polynomial of degree $r = n - k$.

$$g(x)=g_0+g_1x+\cdots+g_rx^r$$

The generator matrix for the systematic code is:

$$\mathbf{G}=\left[\mathbf{I}\middle|\mathbf{P}\right]$$

where **I** is an $k \times k$ identity matrix while **P** is a $k \times r$ matrix. Every row of the **G** matrix is a code word. The i^{th} row is the following code word:

$$\mathbf{g}_i=\left(0,0,\cdots,0,1,0,\cdots,0,\ p_{i,r-1},\ p_{i,r-2},\cdots,\ p_{i,0}\right)$$

where the first *k* components are all zero except the i^{th} component which is 1.

The polynomial corresponding to $\mathbf{g}_i$ is

$$g_i(x)=x^{n-i}+p_{i,r-1}x^{r-1}+p_{i,r-2}x^{r-2}+\cdots+p_{i,0}$$

Since it is a codeword, it must be a multiple of the generator polynomial *g*(*x*). So, we can write:

$$g_i(x)=x^{n-i}+p_{i,r-1}x^{r-1}+p_{i,r-2}x^{r-2}+\cdots+p_{i,0}=a(x)g(x)$$

The generator polynomial *g*(*x*) is a polynomial of degree *r*. We conclude that (over GF(2)):

$$p_{i,r-1}x^{r-1}+p_{i,r-2}x^{r-2}+\cdots+p_{i,0}=x^{n-i}\bmod g(x)\qquad 1\le i\le k \tag{26}$$

Example:

Find the generator matrix for the (7,4) code generated by $g(x) = x^3 + x^2 + 1$.

$$i = 1;\ n-1 = 6;\ x^6 \bmod x^3 + x^2 + 1 = x^2 + x$$
$$i = 2;\ n-2 = 5;\ x^5 \bmod x^3 + x^2 + 1 = x + 1$$
$$i = 3;\ n-3 = 4;\ x^4 \bmod x^3 + x^2 + 1 = x^2 + x + 1$$
$$i = 4;\ n-4 = 3;\ x^3 \bmod x^3 + x^2 + 1 = x^2 + 1$$

so the **P** matrix is:

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

and the generator matrix **G** is:

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

The parity check matrix can be derived as shown in page 25: $\mathbf{H} = \begin{pmatrix} \mathbf{P}^T & \mathbf{I} \end{pmatrix}$.

Non Systematic code

Another simple generator matrix can be derived in non-systematic form. The polynomial $g(x)$ is a codeword. It is of degree r. We can use it as the first row of the generator matrix. The other $k - 1$ rows of **G** will be cyclic shifts of the first row.

$$\mathbf{G} = \begin{pmatrix} 0 & \cdots & 0 & 0 & g_r & g_{r-1} & \cdots & g_1 & g_0 \\ 0 & \cdots & 0 & g_r & g_{r-1} & \cdots & g_1 & g_0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_r & g_{r-1} & \cdots & g_1 & g_0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{27}$$

Using the previous (7, 4) example, we obtain:

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

We can transform the above matrix using elementary row (column) operations in order to derive the parity check matrix. We can also use the generator matrix of the dual code. The generator polynomial $g(x)$ has the following dual code generator:

$$x^k h(x^{-1}) = h_0 x^k + h_1 x^{k-1} + \cdots + h_{k-1} x + h_k$$

Where $h(x)$ is the parity polynomial associated with $g(x)$. So, the parity check matrix is the following $r \times n$ matrix:

$$\mathbf{H} = \begin{pmatrix} 0 & \cdots & 0 & 0 & h_0 & h_1 & \cdots & h_{k-1} & h_k \\ 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{k-1} & h_k & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_0 & h_1 & \cdots & h_{k-1} & h_k & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{28}$$

Specification of codes by roots

We have seen that codewords $c(x)$ are multiples of the generator polynomial $g(x)$. If β is a root of $g(x)$, then β is also a root of $c(x)$. Since $g(x)$ is of degree r, it has r roots. Some of these roots are independent, other are implied (if α is a root, α^2, α^4, … are implied, they generate the same minimal polynomial). For each independent root, we can write

$$c_0\beta^0 + c_1\beta^1 + \cdots + c_{n-1}\beta^{n-1} = 0$$

This equation can be written as:

$$\begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{pmatrix} \begin{pmatrix} \beta^0 \\ \beta^1 \\ \vdots \\ \beta^{n-1} \end{pmatrix} = 0$$

We must remember that the coefficients c_k are elements of GF(2) while the roots belong to an extension GF(2^m) and as such are m dimensional vectors. If we have j independent roots, we can write the above equation for all roots and we obtain:

$$\begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{pmatrix} \begin{pmatrix} \beta_1^0 & \beta_2^0 & \cdots & \beta_j^0 \\ \beta_1^1 & \beta_2^1 & \cdots & \beta_j^1 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{n-1} & \beta_2^{n-1} & \cdots & \beta_j^{n-1} \end{pmatrix} = 0$$

or $$\mathbf{c}\mathbf{H}^T = \mathbf{0}$$

H is then the parity check matrix corresponding to the code.

By transposition, we obtain:

$$\mathbf{H}=\begin{pmatrix}\left(\beta_1^0\right)^T & \left(\beta_1^1\right)^T & \cdots & \left(\beta_1^{n-1}\right)^T\\ \left(\beta_2^0\right)^T & \left(\beta_2^1\right)^T & \cdots & \left(\beta_2^{n-1}\right)^T\\ \vdots & \vdots & \ddots & \vdots\\ \left(\beta_j^0\right)^T & \left(\beta_j^1\right)^T & \cdots & \left(\beta_j^{n-1}\right)^T\end{pmatrix}$$

Example:

The Hamming codes have generator polynomials that are primitive. Any primitive element of the extension field is a root. In this case, there is only one independent root and the parity check matrix is:

$$\mathbf{H}=\begin{pmatrix}\left(\alpha^0\right)^T & \left(\alpha^1\right)^T & \cdots & \left(\alpha^{n-1}\right)^T\end{pmatrix}$$

The (7,4) code has n = 7 and the generator polynomial $g(x)=x^3+x+1$which is primitive. We can use its root to generate GF(2^3) and the parity check matrix will be:

$$\mathbf{H}=\begin{pmatrix}\left(\alpha^0\right)^T & \left(\alpha^1\right)^T & \left(\alpha^2\right)^T & \left(\alpha^3\right)^T & \left(\alpha^4\right)^T & \left(\alpha^5\right)^T & \left(\alpha^6\right)^T\end{pmatrix}$$

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

Bose-Chaudhuri-Hocquenhem Codes (BCH)

A t error correcting BCH code has 2t consecutive powers of an element β as roots in GF(q^m) and it has a length of $q^m - 1$.

Example: In GF(2^m), if t = 1(single error correcting code), we have two roots: β and β^2 and a length of $2^m - 1$. If β is a primitive element, the obtained code is a Hamming code. The generator polynomial is the minimum polynomial of β. Let m = 3, we have $n = 2^3 - 1 = 7$. Consider the field GF(2^3) generated by $p(x)=x^3+x+1$. If $\beta=\alpha$ (which is primitive), the minimum polynomial is the polynomial $p(x)$. Its degree is 3, so, the code is a (7,4) one.

Double error correcting code is more interesting.

Let us consider a primitive double error correcting code. If $\beta=\alpha$(a primitive element), we can select α, α^2, $\alpha^3$and α^4as the roots of its generator polynomial. Among these two, only α and α^3will generate different polynomials. If we transmit $c(x)=c_0+c_1x+\cdots+c_{n-1}x^{n-1}$, we

will receive $r(x) = r_0 + r_1 x + \cdots + r_{n-1} x^{n-1}$. Let us evaluate the received polynomial at the above two roots:

$$r(\alpha) = c(\alpha) + e(\alpha) = e(\alpha) \tag{1.1}$$

$$r(\alpha^3) = c(\alpha^3) + e(\alpha^3) = e(\alpha^3) \tag{1.2}$$

If we have exactly two errors (at positions *i* and *j*), then the error polynomial will have only two coefficients that will be different from zero: $e_i = 1$ and $e_j = 1$, all the other coefficients of $e(x)$ will be zero. So, $e(x) = x^i + x^j$. Evaluating the received polynomial at the roots will provide the following equations:

$$r(\alpha) = s_1 = e(\alpha) = \alpha^i + \alpha^j$$

$$r(\alpha^3) = s_2 = e(\alpha^3) = \alpha^{3i} + \alpha^{3j}$$

Solving for α^j in the first equation and replacing in the second gives:

$$\alpha^j = s_1 + \alpha^i \tag{1.3}$$

$$s_1^2 \alpha^i + s_1 \alpha^{2i} + s_1^3 + s_2 = 0 \tag{1.4}$$

We can replace successive values of α^i in (1.4) and then solve (1.3). The power of α will indicate the location of the error.

Example:

Consider GF(2^4) generated by $x^4 + x + 1$ which is primitive. The field elements are:

0	0000	α^7	1011
α^0	0001	α^8	0101
α^1	0010	α^9	1010
α^2	0100	α^{10}	0111
α^3	1000	α^{11}	1110
α^4	0011	α^{12}	1111
α^5	0110	α^{13}	1101
α^6	1100	α^{14}	1001

Let us design a two error primitive BCH code. We have $t = 2$, so the roots of the generator polynomial are: α, α^2, α^3 and α^4. We use the two independent roots α and α^3. Their minimal polynomials are:

$m_\alpha(x) = x^4 + x + 1$ and $m_{\alpha^3}(x) = x^4 + x^3 + x^2 + x + 1$ giving $g(x) = \text{LCM}\left[m_\alpha(x), m_{\alpha^3}(x) \right]$.

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1$$

So, the code is a (15, 7).

Assume we transmit a code c(x) and we have two errors at $i = 2$ and $j = 13$. The error polynomial is $e(x) = x^2 + x^{13}$. So, $s_1 = \alpha^2 + \alpha^{13} = \alpha^{14}$ and $s_2 = \alpha^{3\times 2} + \alpha^{3\times 13} = \alpha^5$.

$s_1^2 = \alpha^{13}$ and $s_1^3 = \alpha^{12}$. Equation (1.4) becomes:

$$\alpha^{13+i} + \alpha^{14+2i} + \alpha^{12} + \alpha^5 = 0$$

The above equation is satisfied be $i = 2$ and $i = 13$, which are the location of the errors.

A three error correction BCH code can be designed in GF(2^4). It has n = 15 and since $2t$ = 6, we can use as roots: $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5$ and α^6. In this set α, α^3 and α^5 are independent.

We have:

$m_\alpha(x) = (x+\alpha)(x+\alpha^2)(x+\alpha^4)(x+\alpha^8) = x^4 + x + 1$

$m_{\alpha^3}(x) = (x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^{24}) = x^4 + x^3 + x^2 + x + 1$

$m_{\alpha^5}(x) = (x+\alpha^5)(x+\alpha^{10}) = x^2 + x + 1$

This gives $g(x) = \text{LCM}\left[m_\alpha(x), m_{\alpha^3}(x), m_{\alpha^5}(x) \right] = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$

Since $g(x)$ is of degree r = 10, the code is a (15, 5) code.

Syndrome polynomial

In the previous part, we have evaluated the $n - 1$ degree polynomial $r(x)$. We can instead evaluate an $r - 1$ degree polynomial $s(x)$ given by:

$$s(x) = r(x) \bmod \left[g(x) \right]$$

Because $c(x) \bmod \left[g(x) \right] = 0$ and $r(x) = c(x) + e(x)$ then $s(x) = e(x) \bmod \left[g(x) \right]$.

The polynomial $s(x)$ is called the "*syndrome*" polynomial. We use the division hardware shown in Figure 6 to compute $s(x)$. The circuit shown in Figure 6 implements the division of a degree m polynomial by the degree r polynomial $g(x)$ ($g_0 = g_r = 1$). After m shifts, the quotient appears at the output and the remainder is stored in the flip-flops.

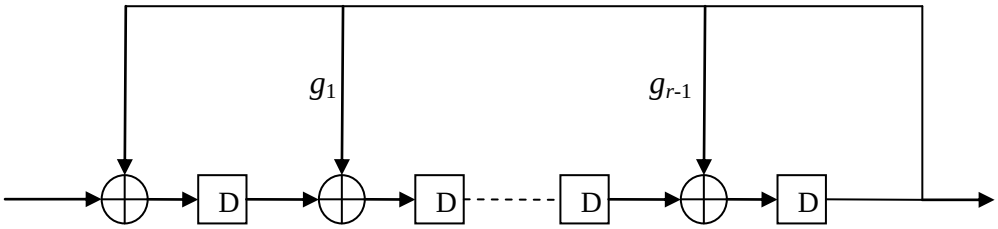


Figure 6 Division of a polynomial by *g*(*x*)

If the input is the received word *r*, the syndrome coefficients will be stored in the flip-flops after *n* clock pulses.

In error detection systems (CRC check), we check whether the division register is zero after we have processed the received bits. Otherwise, we ask for retransmission. In error correction systems, we use the fact that

$$r(\alpha^k) = s(\alpha^k)$$

Where α^k is a root of $g(x)$.

So, in the previous part, instead of evaluating the received polynomial at the different roots of $g(x)$, we evaluate the syndrome polynomial at these roots.