Finite Fields and Entropy in Descrambling

Edmond Rusjan
Dept Math and Science
SUNYIT, Utica, NY,
edmond@sunyit.edu

James P LaRue
dba Jadco Signals, LLC
James@JadcoSignals.com
www.DataPlasticity.com

ABSTRACT

This report presents applications of finite fields to descrambling and to characterizing special sequences.

1. INTRODUCTION

This report was a continuation of a research effort which had as an objective to find an application for Tsallis entropy with modulated bitstreams.

We began by defining a bitstream. A bitstream for our purposes is defined as a structured series of 1's and 0's sent through a randomizer. The structured bitstream (before being sent to the randomizer) is defined as a set of frames of a certain length where in each frame the bits are portioned into a header, a message, and an EDAC. We can think of the header as synchronization tool to indicate that a message is to follow. The EDAC (Error Detection and Correction) is a check on the message.

If one were take a Fourier Transform of this bitstream, spectral harmonics, led in part by the synchronization bits in the header, are easily recognized. And, if this bitsatream was digitally modulated, again, spectral harmonics, led in part by the modulated synchronization bits in the header, are easily recognized.

Modulated bitstreams that yield spectral harmonics are not desired; there is a benefit in power efficiency and bandwidth if these harmonics can be distributed across the bandwidth of the signal. To do that the framed sequence of bits are sent through a randomizer. In particular we have focused our attention on a class of randomizers known as 'feed through randomizer'. In electrical engineering they are known as IIR (Infinite Impulse Response) filters. This is in contrast to FIR (Finite Impulse Response) filters which are used to 'derandomize'.

Randomizers for our application are inherently unstable; this will be shown below. However, randomizers tend to smooth out a spectrum which is good from an engineering design perspective. Randomizers take away spectral harmonics that are present due to the synchronization bits in the header. From an entropy point of view, a (naïve) entropy measure assigned to the structured bits (using a histogram of bits) will be low compared the measure assigned to the randomized bits. There are, of course, theoretical limits on this (naïve) entropic measure which have to do with the most general, albeit, unpractical, class of bitstreams, the uniformly distributed system of bits. We mention the word unpractical since systems of pseudo random uniformly distributed bits are better identified with 'encrypted' bitstreams, and we are not interested in these systems.

Another word for randomizer is scrambler, and we will stick to using this latter expression. Our need to understand scramblers is based on our need to develop an entropy measure on modulated bitstreams, which commonly use scramblers to increase spectral efficiency, in some sense. Scramblers do a good job in removing elements of an obvious structure in the bit domain and pass it on to the modulated domain. Hence our goal was to first understand the characteristics of these scramblers and how they change the structured characteristics of the framed bits and then ultimately identify those characteristics in the modulated domain. Going into this research we knew that the naïve method of entropy measure, that being taking subsets of contiguous bits as received and applying a Shannon type entropy formulation based on the histogram of these contiguous bits, would not be sufficient. In fact, grad student E.J. Yoerger from the University of New Orleans, provided Special Signals with a Matlab based GUI that demonstrated the scramblers ability to hide all traces of the underlying structure of the framed bitstream after it goes through any scrambler. It should be noted that our naïve measure of entropy is very close to the measure associated with Sigmage.

Co-currently, we looked at applying a blind deconvolution method to the scrambled bitstreams to see if, in a deterministic way, we could identify the scrambler itself. This line of research was built on the research also provided to Special Signals by Dr. George Smith of NRL (Naval Research Lab). Dr. Smith took the blind deconvolution method and used it to do blind descrambling (in the bit domain) and blind demodulation (in the modulated domain). With blind descrambling he found that the approach is difficult due to linear algebra constraints which we will talk about later in this paper. However, his research helped guide the effort in this paper to produce two working algorithms; the first algorithm is based on a blind EDAC polynomial assessment while the second algorithm shows the existence of a solution to a system of equations with truncated variable inputs.

In forming a solid base for this extension we leveraged ourselves with the basics of Galois theory and how it pertains bit encoding for EDAC's and we research the basics of DE's (difference equations) and how it pertains to the feedback process found in scramblers. We found a connection between roots of (Galois Field) polynomials where we took some liberty in jumping between Galois fields and the complex number system, and roots of the z-transform of the DE's.

The remainder of this report will be broken up into the following sections. The first section will be the reprint of the summer report with some changes where is will be noted that it is tied to the extended research. The second section will discuss the basics of DE's and how scramblers can be thought of in this domain. We will include two examples: one is taken from the Fibonacci sequence and the other from a scrambler associated with a V.27 modem. The third section will turn to the matrix methods we have developed along the lines of blind identification of EDAC polynomials and scramblers. In the final section we will combine the DE method with the matrix method and examine how we shall proceed with developing a meaningful measure of entropy to apply to these systems in both the bit and modulated domains.

2. THE STRUCTURE OF FINITE FIELDS

Finite fields are fields with a finite number of elements. They were first studied by Evariste Galois and are also called Galois fields. They have been known initially for their mathematical beauty and applications within mathematics. More recently they have found important applications in communications engineering, where they provide the mathematical foundation of coding and scrambling [2]. In coding, they are used as a design tool for Bose-Chaudhuri-Hocquenghem and Reed-Solomon codes. In scrambling, they provide the means to understand the properties of maximal length linear feedback shift register sequences .

Informally, a field is a set in which we can do the same operations as with fractions, real numbers, or complex numbers. More precisely, a field is a set of elements F , including 0 and 1, together with two binary operations: addition + and multiplication $\times$, which are associative and commutative and where multiplication distributes over the addition in the usual way. Every field element u , has a unique negative $-u$, such that $u+(-u)=0$. Every nonzero field element u has a unique reciprocal u^{-1}, such that $u \times u^{-1} = 1$. Therefore, formally, a field is the triple $(F,+,\times)$.

Another way to define a field is as a commutative ring where every nonzero element has a multiplicative inverse. Note that the set of field elements with the operation of addition forms an Abelian (commutative) group and that the set of nonzero field elements with the operation of multiplication also forms an Abelian group.

The **order** of a field is the **number of elements** in the field. If the order is infinite, we call the field an infinite field. For example, rational numbers Q , real numbers R and complex numbers C are all infinite fields.

2.1 Examples

Fields with finite order are called finite fields. For example, Z_2 is a finite field consisting of only two elements 0 and 1, where addition and multiplication are defined modulo 2. To stress that it is a field, we denote it F_2 , i.e., the Galois field with 2 elements. This is the smallest field.

2.1.1 The Field F_p

Similarly, if p is a prime, the Galois field F_p is the finite field of order p , where addition and multiplication are defined modulo p . On the other hand, Z_4 with modulo 4 addition and multiplication is not a field, because the element 2 does not have an inverse.

2.1.2 The Field F_{2^2}

There exists a field with 4 elements. We construct it in an abstract way, referring to the field axioms. Being a field, F_{2^2} has to contain 0 ,1 and two more elements. Let us denote the third element α . Then the field axioms, in particular the closure of addition and the closure of multiplication, imply that $\alpha+1$ and α^2 are also elements of the field. This gives five elements, so at least two of them must be equal to each other. It is easy to check that the only consistent choice is $\alpha+1=\alpha^2$ and that the addition and multiplication operations are defined by the Table 2.1 and Table 2.2, respectively.

Note that the notation is better expressed if we view the fourth element as $\alpha+1$, while the multiplication is more natural with α^2 . Of course, $\alpha+1$ and α^2 are the same element. The correspondence between the additive representation and the multiplicative representation is expressed in the logarithmic Table 2.3.

Table 2.1. Addition in F_{2^2}

$+$	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

Table 2.2. Multiplication in F_{2^2}

$\times$	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

Table 2.3. Logarithms in F_{2^2}

0	0
1	1
α	α
$\alpha+1$	α^2

Note that logarithmic Table 2.3 is sufficient for understanding the field and that the addition and multiplication tables can easily be reconstructed from it. In particular, the addition is simply vector addition and the multiplication of nonzero elements is cyclic.

It is also worth observing, that the equality $\alpha+1=\alpha^2$ implies that α and α^2 are roots of the polynomial x^2+x+1. The field F_{2^2} can be viewed as the polynomial remainders modulo the polynomial x^2+x+1.

2.1.3 *The Field F_{p^m} of Polynomial Remainders*

If $f(x)$ is an **irreducible polynomial** (cannot be factored) of degree m with coefficients in F_p, then the set of remainders (residue classes) modulo $f(x)$ is a finite field of order p^m, denoted F_{p^m}. This is the most important example, because it turns out that all the finite fields are isomorphic to F_{p^m}. We will discuss this example in detail later.

2.2 Brief Summary

Here we present a brief summary of results on finite fields, which are important for coding and scrambling. The rest of the section will explain the concepts in more detail.

2.2.1 *Existence and Uniqueness*

For each prime p and positive integer $m>1$, there exists a finite field F_{p^m} with p^m elements and there exists no finite field with q elements, if q is not a prime power. Any two fields with p^m elements are isomorphic.

2.2.2 *Construction*

The integers modulo p form a prime field F_p under modulo p addition and multiplication. The polynomials $F_p[x]$ over F_p modulo an irreducible polynomial $g(x)\in F_p[x]$ of degree m form a finite field with p^m elements under mod-g(x) addition and multiplication. For every prime p, there exists at least one irreducible polynomial $g(x)\in F_p[x]$ of each positive degree $m\geq 1$, so all finite fields may be constructed in this way.

2.2.3 *Structure*

Under addition, F_{p^m} is isomorphic to the vector space $(F_p)^m$. Under multiplication, the nonzero elements of F_{p^m} form a cyclic group

$$\{1,\alpha,\alpha^2,\ldots,\alpha^{p^m-2}\},$$

generated by a primitive element $\alpha\in F_{p^m}$.

The elements of F_{p^m} are the p^m roots of the polynomial:

$$x^{p^m}-x\in F_{p^m}$$

The polynomial $x^{p^m}-x$ is the product of all monic irreducible polynomials $g(x)\in F_p[x]$ such that $\deg(g(x))$ divides m. The roots of a monic irreducible polynomial $g(x)\in F_p[x]$ form a cyclotomic coset of $\deg(g(x))$ elements of F_{p^m} which is closed under the operation of raising to the p-th power.

For n that divides m, F_{p^m} contains a subfield with p^n elements.

2.3 The Additive Structure

The field F_{p^m} can be viewed as a m dimensional vector space over F_p. Therefore, the addition is simply the vector, i.e., component-wise, addition.

More formally, $(F,+)$ is a finite Abelian group. The Fundamental theorem on Abelian groups, applied to the finite case, states that any finite Abelian group is isomorphic to the direct sum (product) of building blocks $Z(q)$, with $q = p^m$ and p a prime:

$$Z(q) \times Z(r) \times \cdots \times Z(t)$$

For example, this theorem helps us find all Abelian groups of order 8: the cyclic group $Z(8)$, $Z(2)\times Z(4)$ and $Z(2)\times Z(2)\times Z(2)$. It also implies that there is only one Abelian group of order 6 and that the cyclic group $Z(6)$ must be isomorphic to $Z(2)\times Z(3)$. We will see that this theorem has deep consequences for the structure of finite fields.

2.4 The Multiplicative Structure

The field F_{p^m} is determined by the fact that $(F^*,\times)$ is a cyclic group – a fact which is not obvious, so we briefly explain it. First, we need some tools from cyclic groups.

2.4.1 Cyclic Groups and Subgroups

A cyclic group is a group with a single generator. Any cyclic group of order n is isomorphic to Z_n. If g is an element of a group G, then the **cyclic group** $C(g)$, **generated by g**, is always a subgroup G and by Lagrange's theorem its order divides the order of G. In the case where $G = Z_n$, the order of $C(m)$ is the least positive integer k, such that $km = 0 \bmod n$, i.e., such that the integer product km is divisible by n. Thus km is the least common multiple of m and n, denoted by $lcm(m,n)$ and the order of $C(m)$ is:

$$|C(m)| = k = \frac{lcm(m,n)}{m} = \frac{n}{\gcd(m,n)}$$

For example, suppose $n = 10$ and $m = 4$. Then $C(4) = \{4,8,2,6,0\}$ and $|C(4)| = 5$ consistent with $|C(4)| = \frac{lcm(4,10)}{4} = \frac{20}{4} = 5$ and $|C(4)| = \frac{10}{\gcd(4,10)} = \frac{10}{2} = 5$.

We see that $C(m) = Z_n$ if and only if m and n are relatively prime.

2.4.2 The Euler Number

The number of integers in the set $\{0,1,2,\ldots,n-1\}$ that have order n (are relatively prime to n) is called the Euler number and is denoted by $\phi(n)$. For example, in Z_{10}, the integers which are relatively prime to 10, are $\{1,3,7,9\}$, so $\phi(10) = 4$. Apart from the four elements of order 10, Z_{10} also has elements of order 1, 2 and 5, because those are the integers that divide10. There is one element of order 1, namely 0, and $C(0) = \{0\} = Z_1$. There is one element of order 2, namely 5, and $C(5) = \{0,5\} = Z_2$. There are 4 elements of order 5, namely 2, 4, 6 and 8 and $C(2) = C(4) = C(6) = C(8) = \{0,2,4,6,8\} = Z_5$.

In general, Z_n has a cyclic subgroup of order d for each positive integer d that divides n, including 1 and n. C_d consists of $\{0, \frac{n}{d}, \frac{2n}{d}, \ldots, \frac{(d-1)n}{d}\}$ and is isomorphic to Z_d. C_d thus contains $\phi(d)$ elements that are relatively prime to d, each of which has order d and generates C_d. The remaining elements of C_d belong also to smaller cyclic subgroups.

For example, Z_{10} has a subgroup $C_5 = \{0,2,4,6,8\}$ with 5 elements. Four of these elements, namely $\{2,4,6,8\}$, are relatively prime to 5 and generate C_5. The remaining element of C_d, namely 0, has order 1.

Since every element of Z_n has a definite order d that divides n, we can express n as a sum over all possible orders d of the number of elements of that order, which is $\phi(d)$. This gives:

$$n = \sum_{d:d|n} \phi(d)$$

All Euler numbers may be determined recursively from this expression. For example, $\phi(1) = 1$, $\phi(2) = 2 - \phi(1) = 1$, $\phi(3) = 3 - \phi(1) = 2$, $\phi(4) = 4 - \phi(1) - \phi(2) = 2$, ….

2.4.3 The Prime Subfield of a Finite Field

Let F_q be a finite field with q elements. We want to show that q has to be a power of a prime p, i.e., $q = p^m$ and we will use the Primary Subfield as the main tool.

Consider the cyclic subgroup $C(1)$ of F_q generated by the unit element 1. By the cyclic group theorem, $C(1)$ is isomorphic to Z_n and its elements may be denoted by $\{0,1,2,\ldots,n-1\}$, with mod-n addition.

By the distributive law in F_q, the product $i*j$ (in F_q) of two nonzero elements in $C(1)$ is simply the sum of ij ones, which as an element of $C(1)=Z_n$ is $ij(\mathrm{mod})n$. Since this is the product of nonzero elements of F_q, by the field axioms $ij(\mathrm{mod})n$ must be nonzero. This will be true if and only if n is a prime number p. Thus $C(1)$ is a subfield of F_q with a prime number p of elements and hence isomorphic to F_p. Thus the elements of $C(1)$, which are called the integers of F_q, may be denoted by $F_p=\{0,1,2,\ldots,p-1\}$. The addition and multiplication of F_q reduce to modulo p addition and multiplication in F_p. The prime p is called the **characteristic** of F_q. Since the p-fold sum of the identity 1 with itself is 0, the p-fold sum of any element $\beta\in F_q$ with itself is also 0, which means $p\beta=0$. In summary, the integers of a field F_q, form the **prime subfield** $F_p\subseteq F_q$ with p elements, where p is the characteristic of F_q.

2.4.4 The Ring of Polynomials over a Field

The set $F[x]$ of polynomials over a field F has all the properties of a field, except that a nonzero polynomial may not have an inverse. Such a structure is called an **integral domain**, that is, a **commutative ring with identity** and **no divisors of zero** and it is a generalization of the ring of integers Z. The set of polynomials $F[x]$ has several other important properties: it is a Euclidean domain, a principal ideal domain and a unique factorization domain.

2.4.4.1 $F[x]$ Is a Euclidean Domain

Euclidean domain simply means that polynomials have the Euclid property, which describes what happens in polynomial division: for any pair of polynomials $f(x),g(x)\in F[x]$, there exist a unique quotient polynomial $q(x)\in F[x]$ and a unique remainder polynomial $r(x)\in F[x]$ with a smaller degree than the degree of the divisor polynomial $g(x)$, that is $\deg(r(x))<\deg(g(x))$, and:

$$f(x)=q(x)g(x)+r(x).$$

2.4.4.2 $F[x]$ Is a Principal Ideal Domain (PID)

In a ring R, an **ideal** I is a sub-ring with the “absorbing” property $RI=I$. It is easy to see that this is the right property that allows forming the quotient ring R/I.

A **principal ideal** is an ideal with a **single generator** g and can be denoted by $<g>$. In other words, $Rg=<g>$. An integral domain in which every ideal is principal is called a **principal ideal domain** (**PID**). For example, the integers Z are a PID. In fact, every Euclidean domain is a PID, because $<f,g>=<\gcd(f,g)>$, as can be easily verified using the Euclid algorithm. Therefore, $F[x]$ is a PID.

We are interested in constructing fields and the construction will involve division by an ideal, so the natural question is “Under what conditions is quotient a field?” The answer: if and only if the ideal is **maximal** (the only ideal containing it is the ring itself). In $F[x]$, maximal principal ideals are generated by **irreducible polynomials** (polynomials which cannot be factored in $F[x]$). This yields a tool for constructing finite fields: divide $F[x]$ by an ideal generated by an irreducible polynomial.

2.4.4.3 $F[x]$ Is a Unique Factorization Domain

By definition, every monic (leading coefficient 1) polynomial is either irreducible, or it can be factored into a product of monic polynomial factors, each of lower degree. In turn, if a factor is not irreducible, it can be factored further. Following this process, we eventually arrive at a product of monic irreducible polynomials. This factorization is unique (up to the order of the factors), so $F[x]$ is a Unique Factorization Domain.

Note that the unique factorization property may not hold, if the coefficients are not from a field. For example, consider $Z_8[x]$ and the polynomial x^2-1. It can be factored in more than one way:

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

2.4.5 Irreducible Polynomials

A polynomial is irreducible (prime), if it cannot be nontrivially factored. Irreducible polynomials are important because they generate finite fields. **All finite fields can be viewed as polynomials where addition and multiplication are defined modulo an irreducible polynomial**. In a sense, the search for finite fields becomes the search for irreducible polynomials. This is analogous to searching for prime numbers and the **Sieve of Aristothenes** can be used in both cases.

For integers, the method goes as follows. Start with a list of integers greater than 1. The first integer on the list is 2, which is prime. Erase all multiples of 2 (even integers). The next remaining integer is 3, which must be the next prime. Erase all multiples of 3. The next remaining integer is 5, which must be the next prime. Erase all multiples of 5. And so forth.

Similarly, to find all prime (irreducible) polynomials in $F_2[x]$, for example, first list all polynomials of degree 1 or more in $F_2[x]$, in order of degree. (Note that all nonzero polynomials in $F_2[x]$ are monic.) No degree 1 polynomial can have a factor, so the two degree 1 polynomials, x and $x+1$, are both prime. Next, erase all degree 2 multiples of x and $x+1$, namely x^2, x^2+x and x^2+1 from the list of degree 2 polynomials. This leaves one prime (irreducible) degree 2 polynomial, namely x^2+x+1. Next, erase all degree 3 multiples of x, $x+1$ and x^2+x+1 from the list of eight degree 3 polynomials, namely the six polynomials: x^3, x^3+x^2, x^3+x, x^3+x^2+x, x^3+1 and x^3+x^2+x+1. The first four obviously contain x as a factor. The last two (and also the second and third) contain $x+1$ as a factor. That is easily seen by observing that any polynomial in $F_2[x]$ with an even number of coefficients has 1 as a root and therefore contains $x+1$ as a factor. After erasing the six reducible polynomials from the eight degree 3 polynomials, the remaining two polynomials x^3+x+1 and x^3+x^2+1 must be prime. This process yields three prime polynomials of degree four: x^4+x+1, x^4+x^3+1 and $x^4+x^3+x^2+x+1$. Continuing in this, we may list all prime polynomials in $F_2[x]$, up to any desired degree. It turns out that there is at least one irreducible polynomial of every degree. This means that there exists a finite field F_{2^m} for every m.

2.4.6 *Primitive Polynomials*

A **primitive element** of a finite field F_q is an element α whose multiplicative order is $q-1$. This means that all the nonzero elements of the field F_q can be written as:

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

An irreducible polynomial, which has a root α which is a primitive element, is called a **primitive polynomial**. While every irreducible polynomial can be used to generate a finite field, primitive polynomials are especially convenient, because they provide an explicit representation of the field elements as powers of the primitive root α.

2.4.7 *The Field F_{2^3}*

Primitive polynomials offer the most convenient tool for constructing finite fields. The field is the set of remainders modulo the generator primitive polynomial, the addition is the vector addition and the multiplication is cyclic. The logarithmic table gives the correspondence between the additive and the multiplicative representations. For the field F_{2^3}, we construct the field with the primitive polynomial x^3+x+1 and the result is presented in Table 2.4.

Table 2.4. Logarithms in F_{2^3}

0
1
α
α^2
$\alpha^3=\alpha+1$
$\alpha^4=\alpha^2+\alpha$
$\alpha^5=\alpha^2+\alpha+1$
$\alpha^6=\alpha^2+1$

The table indeed contains all the elements in F_{2^3}, because $\alpha^7=1$. There is a second primitive polynomial of degree three, namely x^3+x^2+1, so it may appear that there is another field F_{2^3}. However, the two fields obtained by the two polynomials are isomorphic.

2.4.8 *The Field F_{2^4}*

Following the construction of F_{2^3} in the previous section, we construct F_{2^4} using the primitive polynomial x^4+x+1. The result is summarized in Table 2.5. If we used the second primitive polynomial of degree four, namely x^4+x^3+1 instead, we would obtain the same field in a similar manner. However, if we used the polynomial $x^4+x^3+x^2+x+1$, which is irreducible, but not primitive, then powers of a root α would not be sufficient, because the element α is not primitive. Instead, we would have to use both powers and linear combinations, or find another element β, which is primitive.

Table 2.4. Logarithms in F_{2^4}

0
1
α
α^2
α^3
$\alpha^4 = \alpha + 1$
$\alpha^5 = \alpha^2 + \alpha$
$\alpha^6 = \alpha^3 + \alpha^2$
$\alpha^7 = \alpha^3 + \alpha + 1$
$\alpha^8 = \alpha^2 + 1$
$\alpha^9 = \alpha^3 + \alpha$
$\alpha^{10} = \alpha^2 + \alpha + 1$
$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$
$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$
$\alpha^{13} = \alpha^3 + \alpha^2 + 1$
$\alpha^{14} = \alpha^3 + 1$

The table lists all of the elements of F_{2^4}, because $\alpha^{15} = 1$.

2.4.9 *Minimal Polynomials*

Similarly as irreducible and primitive polynomials can be used to construct finite fields, minimal polynomials are useful in constructing binary codes. A **minimal polynomial** of a field element α is the binary polynomial of smallest degree, which has α as a root. It can be computed in terms of conjugates, which we define next.

2.4.9.1 *Conjugates*

The roots of polynomials with binary coefficients occur in conjugates, similarly as complex roots of polynomials with real coefficients occur in conjugate pairs. For example, the polynomial $z^2 + 1$ has complex roots i and $-i$. Note that the same polynomial, usually written as $x^2 + 1$, when considered with binary coefficients, has a double root $x = 1$, so we need to consider an irreducible polynomial with binary coefficients and there is exactly one such polynomial: $x^2 + x + 1$. It has roots α and α^2 in $GF(2^2)$. The analogy with complex roots end here, though, because while complex conjugate roots always appear in pairs, Galois extension field conjugates may appear in larger sets. If β is a field element of $GF(2^m)$, then the conjugates of β are:

$$\beta, \beta^2, \beta^4, \beta^8, \dots, \beta^{2^{r-1}}$$

where r is the smallest integer such that $\beta^{2^r} = \beta$. For example, the conjugates of α^3 in $GF(2^3)$ are $(\alpha^3)^2 = \alpha^6$ and $(\alpha^3)^4 = \alpha^{12} = \alpha^5$. The conjugate elements in Galois fields $GF(2^2)$, $GF(2^3)$ and $GF(2^4)$ are presented in Table 3.1, Table 3.2 and Table 3.3, respectively.

Table 3.1. Conjugate elements in $GF(2^2)$

Conjugates	Order
1	1
α, α^2	3

Table 3.2. Conjugate elements in $GF(2^3)$

Conjugates	Order
1	1
$\alpha, \alpha^2, \alpha^4$	7
$\alpha^3, \alpha^5, \alpha^6$	7

Table 3.3. Conjugate elements in $GF(2^4)$

Conjugates	Order
1	1
$\alpha, \alpha^2, \alpha^4, \alpha^8$	15
$\alpha^3, \alpha^6, \alpha^9, \alpha^{12}$	5
α^5, α^{10}	3
$\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}$	15

Note that conjugate elements have the same order and therefore if an element is primitive, all its conjugates are also primitive. Recall that the order of an element divides $2^m - 1$ and so if $2^m - 1$ is prime, the field elements will have order $2^m - 1$ and be primitive. For example, all elements of $GF(2^3)$ have order 7 and are primitive. On the other hand, some elements of have order less than 15 and are non-primitive.

One of the properties of conjugates is that they provide a mechanism for going from an extension field to its base field.

Similarly as $(z-i)(z+i)=z^2+1$ yields a real polynomial, $(x+\alpha)(x+\alpha^2)(x+\alpha^4)=x^3+x+1=m(x)$ is a binary polynomial. Note that the product contains a factor for each conjugate. The polynomial $m(x)$ is the binary polynomial of smallest degree, which has α as a root and is called the minimal polynomial of α. Note that all conjugates have the same minimal polynomial.

2.4.9.2 Examples of Minimal Polynomials
Minimal polynomials can be computed for each conjugate class by multiplying together factors corresponding to the conjugates. Results for $GF(2^2)$, $GF(2^3)$ and $GF(2^4)$ are presented in Table 3.4, Table 3.5 and Table 3.6, respectively.

Table 3.4. Minimal Poynomials in $GF(2^2)$

Conjugates	Minimal Polynomial
0	x
1	$x+1$
α, α^2	x^2+x+1

Table 3.5. Minimal Polynomials in $GF(2^3)$

Conjugates	Order
0	x
1	$x+1$
$\alpha, \alpha^2, \alpha^4$	x^3+x+1
$\alpha^3, \alpha^5, \alpha^6$	x^3+x^2+1

Table 3.6. Minimal Polynomials in $GF(2^4)$

Conjugates	Order
0	x
1	$x+1$
$\alpha, \alpha^2, \alpha^4, \alpha^8$	x^4+x+1
$\alpha^3, \alpha^6, \alpha^9, \alpha^{12}$	x^2+x+1
α^5, α^{10}	$x^4+x^3+x^2+x+1$
$\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}$	x^4+x^3+1

3. CODING
Suppose that we wish to transmit a sequence of binary digits across a noisy channel. If we send a 1, a 1 will probably be received. If we send a 0, a 0 will probably be received. Occasionally, however, the channel noise will cause a transmitted 1 to be mistakenly interpreted as a 0 or a transmitted 0 to be mistakenly interpreted as a 1 [Berlekamp]. The goal of coding is to reduce the undesirable effect of the noisy channel, i.e., to detect and correct the channel errors. This is achieved via redundancy.

3.1 Block Codes
In a block code, we take a message of k digits, which we wish to transmit, annex to them r check digits and transmit the entire block of n = k + r channel digits. Assuming that the channel noise changes sufficiently few of these n transmitted channel digits, the r check digits may provide the receiver with sufficient information to enable her to detect and correct the channel errors.

3.1.1 Binary Block Codes
Binary block codes are block codes with digits 0 and 1. Codewords in binary block codes are n-tuples of zeros and ones, and therefore elements of $GF(2)^n$.

3.1.2 The Hamming Geometry
Hamming defined the notions of length and distance in $GF(2)^n$ which, from the point of view of coding, are more natural than the Euclidean length and distance, and still make $GF(2)^n$ a metric space.

3.1.2.1 The Hamming Weight
For every $x \in GF(2)^n$, the Hamming weight:

$$w_H(x) = NumberOfOnesIn(x).$$

The Hamming weight has all the properties of a norm:

a) strict positivity: $w_H(x) \geq 0$, with equality if and only if $x=0$;

b) symmetry: $w_H(-x)=w_H(x)$ (since $-x=x$); and

c) triangle inequality: $w_H(x+y) \leq w_H(x)+w_H(y).$

Therefore, $(GF(2)^n, w_H)$ is a norm space.

3.1.2.2 The Hamming Distance
Every norm induces a metric or distance by considering the norm of a difference and so Hamming weight (norm) w_H defines the Hamming distance $d_H(x,y)=w_H(x-y)$. The Hamming distance has all the properties of a metric, so $(GF(2)^n, d_H)$ is a metric space, called the Hamming space.

Especially important is the minimum Hamming distance d between a pair of codewords, because it determines how many bit errors can be corrected or detected. In particular, at most d-1 bit errors can be detected, or at most (d-1)/2 bit errors can be corrected.

3.1.3 Simple Examples

Among the simplest examples of block codes are the repetition code, the simple parity check code and the ISBN code. We will use them later to illustrate concepts and to construct more complicated codes.

3.1.3.1 The Repetition Code

Simple examples illustrating the redundancy principle are the **repetition codes,** which have $k=1$, r arbitrary, and $n=r+1$, and contain only two code-words: the sequence of n zeros and the sequence of n ones. The first digit is the **message digit** and the rest are the **check digits**. The decoder might use the simple majority rule to decode: count the number of zeros and the number of ones in the received bits; if there are more zeros than ones, decide that the all-zero codeword was sent; if there are more ones than zeros, decide that the all-one codeword was sent; if the number of zeros equals the number of ones, do not decide, i.e., declare a decoding failure. This decoding algorithm will decode correctly in all cases where the channel noise changes less than half of the digits. If the channel noise changes more than half of the bits, the decoder will commit a decoding error.

It is possible to make tradeoffs between decoding errors and decoding failures by modifying the decoding algorithm. For example, an extremely cautious decoder might decode the all-zero word into itself, the all-one word into itself and fail to decode in all other cases. Such an algorithm would detect more errors (at the cost of correcting none) and might be appropriate in cases where a decoding error would result in a disaster.

If the block length is sufficiently large, the repetition code succeeds in making the probability of decoding error arbitrarily small. This comes at a price, which is inefficiency: the information rate is only $R=\frac{1}{n}$. We are usually interested in codes with higher information rates.

3.1.3.2 The Single Parity Check

Extreme examples of high-rate codes are **single parity check** codes, which contain only one check digit. Usually the check digit is chosen so that it results in an overall even parity.

3.1.3.3 The ISBN Code

The International Standard Book Number (ISBN) code, which is used on nearly all recently published books, is a block code which is not binary [Roman]. An ISBN has 10 digits. The first digit indicates the language of the book, the next three digits denote the publisher, the next five digits represent the book number, assigned by the publisher, and the last digit is a redundant check digit, designed to detect errors. The check digit is the solution of the equation:

$$x_1+2x_2+3x_3+\cdots+10x_{10}=0$$

in the field $GF(11)$.

In this report we focus on binary block codes, i.e., codes with digits 0 and 1, because we are interested in applications in communications over binary channels.

3.1.4 Hamming Codes

Hamming was the first coding theorist whose work attracted widespread interest [Early Papers]. In 1950, he constructed a family of **single error correcting** codes, that is, codes with minimum Hamming distance $d=3$, by combining even parity checks over selected information positions.

3.1.4.1 Two Parity Checks

We first illustrate the Hamming construction with the case of 2 parity check bits, i.e., with $r=2$. In the case of no errors, both parity check bits will be zero. If either parity check is nonzero, that is a symptom of an error. Motivated by medical diagnostic terminology, the collection of all (in this case two) the symptoms, is called a syndrome. With two parity check bits, there are 3 nonzero syndromes. Therefore, the syndrome contains enough information to locate a single bit error in a three bit word, so $n=3$ and $k=1$. In other words, we are talking about the three bit repetition code.

Next, we determine over which positions parity checks should act. If the first parity check is applied over the positions 1 and 3 and the second over 2 and 3, then the binary representation of the bit position can be used to indicate whether the bit participates in the specific check. For example, 1=01, so the first bit participates in the first, but not the second parity check. Similarly, 2=10, so the second bit participates in the second, but not the first parity check and 3=11, so the third bit participates in both parity checks.

Finally, we determine which bits are information bits and which are check bits. The check bits can be chosen to be in positions of powers of 2, in this case, the first and second position. The remaining third bit is then the information bit.

3.1.4.2 Three Parity Checks

The $r=2$ case is too simple to justify the reasoning via the Hamming construction, so we need one more example to illustrate the theory and we proceed with the $r=3$ case. Now we have $2^3-1=7$ nonzero syndromes which allow the code-word length $n=7$ and the number of information bits $k=7-3=4$. The check bits are in positions of powers of two: first, second and fourth. The remaining bits, the third, the fifth, the sixth and the seventh, are the information bits. The first parity check is over bits which have a 1 in the least significant bit: 1, 3, 5 and 7. The second parity check is over bits 2, 3, 6 and 7; and the third check is over bits 4, 5, 6 and 7.

To see error correction at work, let us look at an example. Say we want to transmit the information word 1011. First we toss the bits into the information bits slots of the code-word: $\_\ \ \_\ \ 1\ \ \_\ \ 0\ \ 1\ \ 1$, then follow with the first parity check (over bits 1, 3, 5 and 7) to fill the first bit with 0 and obtain $0\ \ \_\ \ 1\ \ \_\ \ 0\ \ 1\ \ 1$, the second parity check (over bits 2, 3, 6 and 7) to fill the second bit with 1:

$0 \quad 1 \quad 1 \quad \_ \quad 0 \quad 1 \quad 1$ and the third parity check (over bits 4, 5, 6 and 7) to fill the fourth bit with 0 and obtain the code-word $0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1$, which we transmit. Suppose the channel noise corrupts the third bit so that $0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1$ is received. The decoder checks the three parity checks and the first two fail, because they contain the corrupted bit, and the third succeeds. The syndrome, which consists of the error symptoms packed from right to left, i.e., from the least significant bit towards the most significant bit, is 011. When read as the binary representation of the number 3, it indicates that the third bit is corrupted. The decoder corrects the corrupted third bit and correctly decodes into $0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1$, from which the message can be read by looking at the information bits 3, 5, 6 and 7. The result is the original message 1011.

3.1.4.3 The General Case

Let r denote the number of check bits. Then the number of syndromes is 2^r and the number of nonzero syndromes is $2^r - 1$. The 0 syndrome represents the state of all received bits being correct. The $2^r - 1$ nonzero syndromes represent all single error locations and we assume no double or higher order errors. We need to have at least n syndromes, in order to represent all the single errors: $2^r - 1 \geq n$. The equality represents the optimal case $2^r - 1 = n$, which maximizes the code length and consequently the code rate. It is not obvious that such codes can be constructed, but Hamming showed they can.

Table 1. Hamming codes

Number of parity checks r	**Length** $n = 2^r - 1$	**Number of message bits = dimension** $k = n - r$	**Number of codewords** 2^k
1	1	0 (useless)	1
2	3	1 (repetition)	2
3	7	4	16
4	15	11	2048
5	31	26	$2^{26} \approx 10^8$
6	63	57	$2^{57} \approx 10^{17}$
…	…	…	…
r	$2^r - 1$	$2^r - 1 - r$	$2^{2^r - 1 - r}$

In addition, Hamming demanded that the syndrome gives the actual position of the error. This implies that every position that has a 1 in its binary representation must be in the first parity check. Thus we see that the first parity check covers position 1,3,5,7,9,11,13,15, …; the second, 2,3,6,7,10,11,14,15,…; the third, 4,5,6,7,12,13,14,15,…; and so on.

Hamming also demanded that codes of different lengths should have message bits in the same positions (and consequently parity check bits in the same positions). If we start filling the codeword with parity check bits and insert message bits, whenever allowed by the $2^r - 1 \geq n$ inequality, then bits 1,2 are parity check bits, bit 3 is a message bit, bit 4 is a check bit, bits 5,6,7 are message bits, bit 8 is a check bit, bits 9,…,15 are message bits, bit 16 is a check bit, etc., and the pattern emerges that bits at powers of 2 locations are check bits and the rest are message bits. This constructively proves the existence of error correcting codes of arbitrary large length.

3.1.5 Maximum Likelihood Decoding

If the decoder receives an n-tuple , which is not a codeword, it attempts to decide, which message was sent. Since each message results in a codeword being transmitted and since under reasonable channel noise conditions a single bit error is a low probability event, a double bit error is less likely than a single bit error, and the probability for an n bit error decreases with n., the most likely codeword to have been sent is the one which is the smallest number of bit errors away from the received n-tuple, that is, the one which is closest, in Hamming distance, to the received n-tuple. In maximum likelihood decoding, the decoder checks the distances to all the codewords and picks the closest codeword. This is conceptually simple, but there is a problem: the number of codewords increases exponentially with the code length, so the complexity of the maximum likelihood is exponential. In other words, the maximum likelihood algorithm is not practical for large codelengths.

This is the central problem in coding and results in the need to impose more structure on codes and then use the structure to develop faster decoding algorithms.

3.2 Linear Codes

Hamming codes impose several even parity checks on the bit sequence to define code-words. In other words, code-words are solutions of a system of linear equations, over $GF(2)$, of course. In the linear algebra language, the code (the set of code-words) is the nullspace of the matrix of coefficients of the system of equations, called the parity check matrix. Note that the nullspace is always a linear subspace (vector subspace) and therefore a linear space (vector space).

It is natural then to relax the constraints of the Hamming construction and study the properties of codes which are linear spaces. Such codes are called **linear codes**. More precisely, linear codes are k dimensional subspaces of R^n.

Linear algebra tells us that the two points of view, parity check matrix and subspace, are complementary. This is because every subspace can be viewed as the nullspace of the projection matrix onto that subspace and the parity check matrix is the projection matrix. Conversely, as already mentioned, the parity check matrix defines its subspace, which is a linear space.

A third point of view is to focus on the encoding process, which is a mapping from R^k into R^n, whose range is the code. This mapping is linear and is an embedding (1-1 mapping). Therefore it can be represented by a matrix, called the **generation matrix** G. The **parity check matrix** is then conveniently called H.

To summarize, if $m \in R^k$ is the message, then the corresponding codeword $c \in R^n$ is given by $c = mG$ and $cH^T = 0$. We follow the coding literature convention, where the transpose of the H matrix is used for convenience – easier formatting of formulas in text, when each row represents one parity check. Furthermore, as in the coding literature, the convention is to represent vectors as rows and put them on the left of the matrices, because then it is easier to think of the corresponding block diagram picture: the message m passes through the block G to become a codeword and then passes through the block H and gets annihilated. Others, for example, Wikipedia, follow the mathematics convention, where matrices are on the left and vectors on the right. Notice that by combining the generation and the check equations, we obtain $cH^T = (mG)H^T = m(GH^T) = 0$, for every $m \in R^k$, which implies $GH^T = 0$.

3.2.1 Examples

The repetition code, the even single parity check code and the Hamming codes are all linear codes, while the odd single parity check code and the ISBN code are not linear. For the three linear codes examples, we construct the generation matrices and the parity check matrices. We also note the extreme cases, where the dimension k is 0 or n.

3.2.1.1 The Trivial, or (n,0), Code

For every length n code, the smallest possible dimension of the code is k=0. This code has only one codeword, namely the 00…0 codeword. This code is called the trivial code and is not useful.

3.2.1.2 The Universe, or (n,n), Code

On the other extreme is the universe code, which has dimension k=n, where the code is the whole space, that is, every word is a codeword. This simply means that all messages are transmitted in their original form, without being encoded.

3.2.1.3 The Even Single Parity Check Code

The code consisting of all n-tuples with an even number of ones is an (n,n-1) linear binary code, called the even-weight of single parity check (SPC) code of length n. Of all the (n,n-1) linear binary codes, the SPC is the most important, so when we say the (n,n-1) code, we will mean the SPC.

3.2.1.4 The Repetition Code

Every linear code contains the 00…0 codeword. The smallest nontrivial codes contain only one other codeword and are dimension 1, or (n,1), codes. In order to maximize the Hamming distance between the two codewords, we choose the nonzero codeword to be 11…1. This is the repetition code of length n. We will reserve the (n,1) notation for the repetition code.

The repetition code can be represented by the generating matrix G:

$$G = [1 \quad 1 \quad 1 \quad . \quad . \quad . \quad 1] = [I_1 \mid P]$$

The corresponding parity check matrix H is:

$$H = [P^T \mid I_{n-1}] = \begin{bmatrix} 1 & 1 & 0 & ... & 0 \\ 1 & 0 & 1 & ... & 0 \\ .. & .. & .. & ... & .. \\ 1 & 0 & 0 & ... & 1 \end{bmatrix}$$

3.2.1.5 The (7,4) Hamming Code

All Hamming codes are linear codes because they are constructed as nullspaces of the parity check matrix H, where each row corresponds to one of the parity checks. If we follow the Hamming construction directly, by putting ones in places which contribute to the parity check and zeros elsewhere, then the parity check matrix H is:

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

The corresponding generating matrix G has columns 3,5,6,7 equal to the columns of the identity matrix I_4, because the 3,5,6,7 codeword bits are the message bits. This partially determines the generating matrix G:

$$G = \begin{bmatrix} \_ & \_ & 1 & \_ & 0 & 0 & 0 \\ \_ & \_ & 0 & \_ & 1 & 0 & 0 \\ \_ & \_ & 0 & \_ & 0 & 1 & 0 \\ \_ & \_ & 0 & \_ & 0 & 0 & 1 \end{bmatrix}$$

The rest of the matrix G can be determined by observing the identity $GH^T = 0$. The 1,2,4 codeword bits are parity check bits, so the 1,2,4 columns of the H matrix are columns of the identity matrix I_3. If we write the remaining bits of each row of the H matrix in the corresponding column of the G matrix, we obtain:

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

This agrees with the definition of the G and H matrices in the Wikipedia, except of course, that G is transposed.

3.2.2 Systematic Codes

It is often convenient to group the message bits together at the beginning or at the end of the codeword. This bit reordering corresponds to reordering of the columns of G and H and yields what is called an equivalent code.

For example, the (7,4) Hamming code written as a systematic code has the generator matrix G:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = [I_4 \mid P]$$

The corresponding H matrix can be obtained simply by:

$$H = [P^T \mid I_3] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

To encode a message, simply multiply it by G. For example, if the message is $m = [1 \quad 1 \quad 0 \quad 0]$, the codeword is:

$$c = mG = [1 \quad 1 \quad 0 \quad 0]G = [1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$$

All the codewords of the (7,4) code can be obtained by multiplying all the 16 messages by the generating matrix G. The result is summarized in Table 2.

Table 2. Hamming (7,4) systematic codewords

Counter	Message m	Codeword c=mG
0	0 0 0 0	0 0 0 0 0 0 0
1	0 0 0 1	0 0 0 1 0 1 1
2	0 0 1 0	0 0 1 0 1 1 0
3	0 0 1 1	0 0 1 1 1 0 1
4	0 1 0 0	0 1 0 0 1 1 1
5	0 1 0 1	0 1 0 1 1 0 0
6	0 1 1 0	0 1 1 0 0 0 1
7	0 1 1 1	0 1 1 1 0 1 0
8	1 0 0 0	1 0 0 0 1 0 1
9	1 0 0 1	1 0 0 1 1 1 0
10	1 0 1 0	1 0 1 0 0 1 1
11	1 0 1 1	1 0 1 1 0 0 0
12	1 1 0 0	1 1 0 0 0 1 0
13	1 1 0 1	1 1 0 1 0 0 1
14	1 1 1 0	1 1 1 0 1 0 0
15	1 1 1 1	1 1 1 1 1 1 1

3.2.3 Distance Invariance

In general, all pairs of codewords need to be checked in order to find the minimum Hamming distance. For linear codes, it is enough to check the Hamming weights, because the minimal Hamming distance equals the minimal Hamming weight: d=w. This is a consequence of the distance invariance property, which we now describe.

For linear codes C, if c is a codeword:

$$c + C = C$$

This means that the code is invariant under translations by codewords. Since $d_H(x, y) = w_H(x - y)$, the distance profile from any codeword (the set of Hamming distances to all the other codewords) equals the distance profile from any other codeword and in particular the distance profile from 0, which is the weight profile.

Geometrically speaking, the code looks the same no matter at which codeword the observer is sitting. Consequently, the minimal Hamming distance equals the minimal Hamming weight: d=w.

3.2.4 Orthogonality in Hamming Space

The concept of orthogonality is defined in Hamming space the same way it is defined in Euclidean space, namely through the inner product – two vectors $\vec{x}$ and $\vec{y}$ are orthogonal, if and only if their inner product is 0:

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i = 0$$

However, orthogonality in Hamming space has several surprising properties. For example, a vector can be orthogonal to itself. As an example, take any vector with an even number of ones. Furtherrmore, the projection theorem does not hold and consequently the Gramm-Schmidt process cannot be used to orthogonalize bases. For example, the (3,2) SPC code C = {000, 011, 101, 110} does not have an orthogonal basis.

3.2.5 Dual Codes

Since the orthogonal complement of a vector subspace is again a subspace, the orthogonal complement of an (n,k) code C is an (n,n-k) code C^{perp}, called the dual code of C. If G is the generator matrix for C, then the rows of G are a basis for C and if H is the parity check matrix for C, then we have seen that $GH^T = 0$. This means that the rows of H are orthogonal to C, so they can be viewed as a basis for C^{perp} and H can be viewed as a generator matrix for C^{perp} and G can be viewed as a parity check matrix for C.

The dual code of the dual code is the original code. The dual code of the universal code is the trivial code and vice versa. The dual code of the repetition code is the single parity check code. Since for n=2, the repetition code is the same as the SPC code, C={00,11}, this code is a self dual code. The dual of the (7,4)

Hamming code is the (7.3) code whose generating matrix is the Hamming (7,4) parity check matrix H.

3.3 Cyclic Codes

Linear codes which are invariant under the cyclic shift are called cyclic codes. In other words, in a cyclic code, a cyclically shifted codeword is again a codeword. Explicitly, for every codeword $c=[c_{n-1} \quad \dots \quad c_1 \quad c_0]$, the (left) shifted codeword:

$$Sc = S[c_{n-1} \quad \dots \quad c_1 \quad c_0] = [c_{n-2} \quad \dots \quad c_0 \quad c_{n-1}],$$

is also a codeword.

For example, the repetition code is obviously cyclic, because each codeword by itself is shift invariant. The Hamming (7,4) is also cyclic, as can be easily checked using Table 2. In particular , the 0th and 15th codewords are invariant, while the rest of the codewords form two length 7 orbits under the shift S:

$$S: 1 \to 2 \to 5 \to 11 \to 6 \to 12 \to 8 \to 1$$

$$S: 3 \to 7 \to 14 \to 13 \to 10 \to 4 \to 9 \to 3$$

An example of a code which is linear but not cyclic is the (5,2) code with the generator matrix:

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

Note that the (left) shift of the codeword $1 \quad 0 \quad 1 \quad 1 \quad 0$ is $0 \quad 1 \quad 1 \quad 0 \quad 1$, which is not a codeword.

3.3.1 *Additional Operation: Convolution*

Cyclic codes enrich the vector space structure of linear codes by adding the operation of cyclic convolution of two vectors. The reason the convolution must be cyclic is that the operation must be internal. The resulting algebra is isomorphic to the ring of binary polynomials of degree n modulo the polynomial $x^n - 1$, viewed as an algebra over $F_2^{\ n}$. We denote this algebra by V_n.

To see why we need to mod out $x^n - 1$, consider multiplying x^{n-1} by x. Without modding out, the result would be x^n, which is not in the algebra, because the degree is too high, and the polynomial multiplication would fail to be an internal operation. After modding out, the polynomial multiplication is an internal operation and it corresponds to the cyclic convolution, if we view polynomials as vectors. This isomorphism between the algebra of polynomials and vectors allows us to identify vectors with polynomials and in the rest of the report we will be switching freely between the two notations.

3.3.2 *Hardware Implementation*

The great practical importance of cyclic codes is that they can easily be implemented in hardware.

In hardware implementations, the polynomial algebra can be implemented with linear shift registers. The is based on the basic capability of the flip-flop element, which is a storage element regulated by an external clock, to implement a delay: the input to the flip-flop appears as its output one unit of time later. Consequently the flip-flop can implement the cyclic shift, i.e., multiplication by x and a series of flip-flops connected into a shift register can represent a polynomial.

The goal of hardware encoding and decoding of cyclic codes will therefore be achieved, if we can show that encoding and decoding can be reduced to polynomial operations. We will do this later in this section.

3.3.3 *Cyclic Codes as Ideals*

A linear code C is cyclic if and only if it is an ideal in V_n. At first sight it may appear that it is harder to be an ideal than a cyclic code, because if the product of any codeword and any element (polynomial) in V_n is in V_n, then in particular the product of a codeword and x is in V_n. However, the two notions are indeed equivalent, because multiplication by any polynomial can be built from repeated multiplication by x and linearity.

All ideals (cyclic codes) in V_n are **principal**, i.e., generated by a **single generator polynomial** $g(x)$, which is the minimal degree nonzero codeword. Therefore, we can say that the code is generated by g and write $C=(g)$. Since $g \in C$, it is obvious that the ideal generated by $g(x)$ is contained in C, i.e., $(g) \subseteq C$. To see that the converse is true, use Euclid's property, i.e., if s is another codeword $s \in C$, then divide g into it: $s = gf + r$, where the degree of r is smaller than the degree of g. Since s and g are codewords, so is r. Therefore, $r = 0$, or else g cannot be the minimal degree nonzero codeword. From $s = gf$ we see that g divides s and therefore $C=(g)$. It is easy to see that the generator polynomial g is unique, if we require that it is monic (leading coefficient 1), for if there were another g' such that $C=(g')$, then g and g' would have to divide each other and that is only possible if they are equal.

We will show later that the Hamming (7,4) code is generated by the polynomial $g(x) = x^3 + x + 1$.

3.3.4 *Factoring* $x^n - 1$

The generator polynomial g divides $x^n - 1$ [2]. This follows from Euclid's property $x^n - 1 = gs + r$, which by modding out $x^n - 1$ implies $0 = gs + r$ in V_n. Therefore the remainder r is a codeword and as such it must be zero, or else it would be the minimal degree nonzero codeword. This means that V_n is never an integral domain and therefore never a principal ideal domain, because g is always a zero divisor.

In the Hamming (7,4) example, $g(x) = x^3 + x + 1$ indeed divides $x^7 - 1$ and the quotient is $h(x) = x^4 + x^2 + x + 1$.

In order to find all cyclic codes, all we need to do is find all divisors of $x^n - 1$. Let $x^n - 1 = g_1 g_2 \ldots g_t$ be the complete factorization of $x^n - 1$ into irreducible polynomials over F_2.

For odd n, $x^n - 1$ is square free and the factors g_i are all distinct. To see this, notice that $x^n - 1$ and its derivative do not have any common factors for odd n. The cyclic codes $C_i = (g_i)$ generated by g_i are maximal ideals in V_n and are called **maximal cyclic codes**.

3.4 Bose-Chaudhuri-Hocquenghem (BCH) Codes

In cyclic codes, code-words have their generating polynomial as a factor. Therefore, the roots of the generator polynomial are the roots of the code-words and it is natural to reconsider the cyclic codes in terms of roots in an extension field - Galois field. Furthermore, the Galois field techniques can be used as a design tool to construct codes with a prescribed minimal Hamming distance. In particular, the **Bose-Chaudhuri-Hocquenghem** (**BCH**) codes are **designer t-error correcting** cyclic codes with minimal Hamming distance $2t + 1$ constructed using roots in finite fields.

3.4.1 The Roots of the Hamming Code Generators

The simplest Hamming code is the (3, 1) repetition code, which has the generator polynomial $g(x) = x^2 + x + 1$ with roots α and α^2 in $GF(2^3)$. Note that the generator polynomial g can be factored into $g(x) = (x + \alpha)(x + \alpha^2)$.

The generator polynomial $g(x) = x^3 + x + 1$ of the Hamming (7, 4) code has roots α, α^2 and α^4 belonging to $GF(2^3)$ and can be expressed as $g(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^4)$. We can think of the generating polynomial as being specified by its roots. This suggests the idea of designing cyclic codes by selecting well chosen sets of finite field elements to be the roots of the generator polynomial. Note that the generator polynomial g is the minimal polynomial $m(x)$ of α.

The generator polynomial of the Hamming (15, 11) code $g(x) = x^4 + x + 1 = m(x)$ is the minimal polynomial of α in $GF(2^4)$ and its roots are α, α^2, α^4 and α^8.

To construct larger length Hamming codes we consider larger Galois fields and higher degree minimal polynomials. In particular, we can construct the r check bit Hamming code of length $2^r - 1$ by selecting a degree r generator polynomial as the minimal polynomial of a primitive element α in the field $GF(2^r)$.

3.4.2 The BCH Construction

The BCH construction is a generalization of the method used in the previous section to construct Hamming codes. Instead of taking the generator polynomial to be the minimal polynomial of a single element, BCH takes the least common multiple of minimal polynomials of successive powers of elements. Most often, the successive powers are of the primitive element and then the resulting code is called the primitive BCH code. More specifically, for a t-error correcting BCH code, consider the sequence of powers of a primitive element α in $GF(2^r)$:

$$\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}.$$

Then select the generator polynomial g to be the least common multiple of the minimal polynomials of the elements in the sequence:

$$g(x) = LCM[m_1(x)m_2(x)m_3(x), \ldots, m_{2t}(x)]$$

Note that the even powers in this sequence are redundant, because they have a conjugate which appears earlier in the sequence and that conjugate has the same minimal polynomial. In particular, in the case of Hamming codes, where $t = 1$, it may at first appear that α and α^2 need to be considered, but in fact α suffices.

The code length is determined by the Galois field $GF(2^r)$ and is the same as the length of the Hamming codes, namely $2^r - 1$.

3.4.3 BCH Code Examples

We have already seen that single error BCH codes are Hamming codes. Next, we construct a few examples of double and triple error correcting BCH codes.

3.4.3.1 Double Error Correcting BCH Codes

The simplest double error correcting BCH code requires the sequence of four powers of a primitive element α:

$$\alpha, \alpha^2, \alpha^3, \alpha^4.$$

In $GF(2^3)$ these elements have minimal polynomials $m_1(x) = m_2(x) = m_4(x) = x^3 + x + 1$ and $m_3(x) = x^3 + x^2 + 1$. The least common multiple of these two polynomials is their product, so the generator polynomial is $g(x)=m_1(x)m_3(x)=(x^3+x+1)(x^3+x^2+1)$.

4. LINEAR RECURSIVE SEQUENCES

Linear Recursive Sequences (LRS) over finite fields are the mathematical foundation for scramblers. For the introduction, we first illustrate the more familiar LSRs over real numbers, by the example of Fibonacci numbers. The analysis tools are analogous to solving linear differential equations with constant coefficients.

4.1 Fibonacci Numbers

The sequence of Fibonacci numbers $\{s_t\}$ is defined by the initial condition $s_0=0$, $s_1=1$ and the recurrence relation:

$$s_t = s_{t-1} + s_{t-2} \qquad \text{for} \qquad t \geq 2$$

The repeated application of the recurrence relation yields the well known sequence

5. SCRAMBLING APPLICATIONS

Scrambling is a binary bit-level processing applied to the transmission signal in order to make the resulting binary sequence appear more random [Lee – Scrambling Ch2]. A scrambler can be built from shift registers and exclusive or gates. The corresponding descrambler can be viewed as a time reversed scrambler. This means it has the same structure as the scrambler, but the bit stream passes through it in the opposite direction. The scrambler and the descrambler need to be synchronized in order to function properly. Depending on the synchronization method used, scrambling techniques are classified into three categories: the frame synchronous scrambling (FSS), the distributed sample scrambling (DSS) and the self synchronous scrambling (SSS).

In the FSS, the states of the scrambler and descrambler shift registers get synchronized by being simultaneously reset to the specified states at the start of each frame. In the DSS, samples taken from the scrambler shift registers are transmitted to the descrambler in a distributed manner for use in synchronizing the descrambler shift registers. In the SSS, the states of the scrambler and descrambler shift registers are automatically synchronized without any additional synchronization process.

5.1 Frame Synchronous Scrambling

Frame Synchronous Scrambling employs an autonomous system consisting of shift registers and exclusive or gates, which is called the Shift Register Generator (SRG). An SRG can be engineered in such a way to generate a desired Pseudo Random Binary Sequence (PRBS) for use in scrambling. In the scrambling part, the transmission signal is scrambled by adding the PRBS to it and in the descrambling part the same PRBS is added to the scrambled signal for the recovery of the original signal. In order for this to work, the scrambler and descrambler have to be identical and synchronized to the same state. To achieve this the FSS resets the scrambler and descrambler SRGs to some predetermined states at the beginning of each frame. A well known application of an FSS is in SDH/SONET lightwave transmission system. It is composed of seven shift registers and one exclusive or gate. The scrambler and descrambler generate the PRBSs $\{s_k\}$ and $\{\hat{s}_k\}$ of length 2^7-1. The transmission signal $\{b_k\}$ is scrambled by adding the PRBS $\{s_k\}$ generated by the scrambler SRG. The scrambled signal $\{b_k+s_k\}$ is descrambled by adding the PRBS $\{\hat{s}_k\}$, generated by the descrambler SRG, which becomes identical to $\{s_k\}$ when the descrambler SRG is synchronized to the scrambler SRG. Therefore, the descrambled signal $\{b_k+s_k+\hat{s}_k\}$ becomes identical with the original signal $\{b_k\}$ in the synchronized state. For this synchronization, the scrambler SRG state and the descrambler SRG state are both set to "1111111" at the beginning of each SDH/SONET frame.

5.2 Distributed Sample Scrambling

Distributed Sample Scrambling is similar to FSS, which scrambles the signal by adding a PRBS generated by an SRG. The difference between the DSS and FSS lies in the synchronization method. In the DSS, the samples of the scrambler SRG are distributed (transmitted) to the descrambler in parallel to the scrambled signal, where they are used to correct the descrambler SRG state in such a way that it eventually becomes identical to the scrambler SRG state. The samples of the scrambler SRG are usually taken and conveyed over some available slots in the transmission frame in a distributed manner – hence the name. The advantage of DSS over FSS is that in DSS the SRGs are NOT reset at the beginning of each frame and thus the transmission signal is scrambled by a continuous PRBS stream, resulting in superior scrambling. This comes at a cost: in DSS, we need to continuously check whether the descrambler SRGs stay in the synchronous state.

An example of a transmission system using DSS is the cell-based ATM. In this system, the scrambler and descrambler consist of 31 shift registers and one exclusive-OR gate and they generate PRBS of length $2^{31}-1$. For synchronization, the samples $\{z_i\}$ of the scrambler SRG state are taken from the PRBS $\{s_k\}$ and distributed to the descrambler over the Header Error Control (HEC) field of the ATM 53 bit cell. The descrambler generates its own samples $\{\hat{z}_i\}$ of the SRG state in the same manner and compares them to the transmitted ones. If the two sets of samples are identical, no action takes place. If they are different, a correction logic changes the descrambler SRG state and brings it in sync with the scrambler SRG state in at most 31 iterations.

5.3 Self Synchronous Scrambling

Self Synchronous Scrambling is quite different from FSS and DSS. In SSS, the signal, instead of being added to the PRBS generated by the SRG, passes directly through the SRG. The scrambled system gets descrambled as it passes through an input/output reversed replica of the scrambler. In this operation, the transmission signal itself controls the state of the shift registers in the scrambler and the scrambled signal controls the state of the shift registers in the descrambler. The scrambler and descrambler are automatically synchronized once the number of received bits reaches the shift register length. The term self-synchronous refers to this synchronization operation.

An example of a transmission system using SSS is the SDH-based ATM. The ATM cell stream $\{b_k\}$ gets scrambled as it passes through the 43 shift registers in the scrambling part and the scrambled signal $\{\hat{b}_k\}$ 9s descrambled as it passes through the input/output reversed shift register block in the descrambler. The states of the shift registers in the descrambler become automatically synchronized to those in the scrambler after the reception of the first 43 scrambled bits.

The advantage of SSS is that it achieves good scrambling performance for both short and long frames. This comes at a price: a single bit error occurring in scrambled data due to a transmission bit error typically causes a multi-bit error, or error multiplication in the descrambled bit stream.

5.4 Scrambling Sequences

As mentioned earlier, the function of scrambling is to randomize the bit stream before transmission. Therefore, the scrambling sequence $\{s_k\}$ should be such that the scrambled signal $\{b_k + s_k\}$ is sufficiently random. If the scrambling sequence $\{s_k\}$ itself is random, then the scrambled signal $\{b_k + s_k\}$ is also random. However, in practice, the scrambling sequence is generated by a SRG, so it is not random. Therefore, it is desirable to investigate the properties associated with randomness and to use such propertied in the design of the scrambling sequence $\{s_k\}$.

5.4.1 Randomness Properties

The following properties of binary random sequences are particularly useful in the design of scrambling sequences: nearly equal number of 0s and 1s, long runs appear less frequently than short runs and the autocorrelation function is close to the delta function. More precisely, one half of the runs are length 1, one quarter of the runs are length 2, one eighth of the runs are length 3, etc., decaying exponentially, until we reach the longest run, which has frequency 1. Sequences possessing these properties are called Pseudo Random Bit Sequences (PRBS).

5.4.2 Shift Register Generators

Shift Register Generators (SRG) are widely used to generate PRBSs . SRGs consist of shift registers and exclusive-OR gates. Two types are commonly used in FSS: Simple SRG (SSRG) and Modular SRG (MSRG). They differ in how the feedback is implemented. Loosely speaking, they have arrows pointing in the opposite direction. In the SSRG, the arrows go out from the shift register stack into the OR gates, while in the MSRG, they are reversed. More precisely, in SSRG, the state $\{d_{L-1,k}\}$ of the last shift register is controlled by the other shift registers. In contrast, in the MSRG, the state of the last shift register $\{d_{L-1,k}\}$ controls the state of other shift registers. In fact the SSRG and MSRG are the simplest types among many other possible SRGs.

The scrambling sequences and the relevant SRGs are the most important parts to understand to properly analyze and synthesize scramblers, including the FSS, the DSS and even the SSS. The unified description of the three categories of scramblers (FSS, DSS and SSS) is provided by the concept of the Sequence Space, which we discuss next.

5.5 Sequence Spaces

The concept of Sequence Space enables a rigorous definition and description of sequences and SRGs. Loosely speaking, a Sequence Space is a vector space of sequences satisfying the relation specified by a characteristic polynomial. More precisely, for a binary coefficient polynomial $P(x) = \sum_{i=0}^{L} c_i x^i$, we define the Sequence Space as the set of all sequences: $V[P(x)] = \{\{s_k, k = 0,1,\ldots\} : \sum_{i=0}^{L} c_i s_{k+i} = 0, k = 0,1,\ldots\}$ satisfying the polynomial relation. The addition and scalar multiplication of sequences is defined in the obvious way, componentwise, The polynomial $P(x)$ characterizing the sequence space is called the characteristic polynomial of the sequence space $V[P(x)]$.

Since the defining relation of the sequence space can be viewed as a recurrence relation $s_k = \sum_{i=1}^{L} c_i s_{k+i}, k = 0,1,\ldots$ or $s_{k+L} = \sum_{i=0}^{L-1} c_i s_{k+i}, k = 0,1,\ldots$, it is clear that each sequence is determined by the first L elements, which we call the initial vector of the sequence. Therefore, the sequence space $V[P(x)]$ is L dimensional and has 2^L elements. In other words, the dimension of the sequence space equals the degree of the characteristic polynomial.

5.5.1 Elementary Basis

Sequences E^i, whose initial vector is $e_i = (0,\ldots,0,1,0,\ldots,0)$, the i-th elementary basis vector of the initial space are called elementary sequences. and similarly for $e_2 = (0,1,\ldots,0), e_3, \ldots, e_L$. The set of the L elementary sequences $E^1, \ldots, E^L$ forms a basis for $V[P(x)]$, called the elementary basis.

5.6 The Trace and Norm in a Field

The trace and norm can be defined in any field, but are particularly useful tools in finite fields. In the field of complx numbers, the trace of z is simply twice the real part:

$$Tr(z) = z + \bar{z}$$

And the norm is the modulus squared:

$$N(z) = z\bar{z}$$

Note that both the trace and the norm are real numbers and that the set of real numbers is a subfield of complex numbers. The trace and the norm can therefore be viewed as compressing information from the extension field to the subfield.

For finite fields, the trace and the norm compress information in a similar way. A finite field F_q can be viewed as a subfield of F_{q^n} and both the trace and the norm as functions from F_{q^n} to F_q :

$$Tr^{F_{q^n}}{}_{F_q}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \ldots + \alpha^{q^{n-1}}$$

$$N^{F_{q^n}}{}_{F_q}(\alpha) = \alpha\alpha^q\alpha^{q^2}\ldots\alpha^{q^{n-1}}$$

In other words, both for complex numbers and for finite fields, the trace and is the sum of the conjugates and the norm is the product of the conjugates.

6. DIFFERENCE EQUATIONS

A linear, constant-coefficient Nth order difference equation (DE) for Single-Input-Single- Output (SISO) systems can be written as

$$y(n)+a_1y(n-1)+\ldots a_Ny(n-N)=b_0x(n)+b_1x(n-1)+\ldots b_Lx(n-L)$$

, (Eq. 6. 1)

where N accounts for the N possible delays of the output and L accounts for the L possible delays of the input. The system is called recursive when previous values of the output enter into the calculation of the most recent output, y(n). The system is called nonrecursive when there are no previous outputs used in the calculation of y(n), leaving the updating to y(n) as a function of the x(i) inputs only. The order of the DE depends on the number N and not on the number L.

To analyze the stability of the system described by a DE, often the terms on the right of Eq. 1 are set to zeros which leaves us with the output y(n) and its N delays. We form what is called a characteristic equation for a recursive system by substituting in a trial solution $y(n)= z^n$ (or, take the z-transform) into the homogenous equation DE

$$y(n)+a_1y(n-1)+\ldots a_Ny(n-N)= 0$$

(Eq.6.2)

to get

$$z^n + a_1z^{n-1} + a_2z^{n-2} + \ldots + a_Nz^{n-N} = 0.$$

By factoring out z^n we get

$$z^n(1+a_1z^{-1} + a_2z^{-2} + \ldots + a_Nz^{-N}) = 0 \quad \text{(Eq. 6.3)}$$

where we analyse the portion in the parenthesis which is known as the characteristic equation (CE). The N roots from this equation are called the characteristic roots and it is from these roots system stability can be characterized; noting if the a_i are real numbers the roots will appear in conjugate pairs. The system is said to be stable (under certain conditions) is each r_k has magnitude less than one. Also, we may rewrite the solution to Eq. 2 in terms of these roots (and for a given set of initial conditions) in the form:

$$y_{IC}(n)=C_1r_1^n + C_2r_2^n + \ldots + C_Nr_N^n \quad \text{(Eq. 6.4)}$$

Where the subscript *IC* indicates this is the initial condition solution.

Example 1 Fibonacci.

Example 2 V.27 scrambler $1+x^{-6}+x^{-7}$ as noted in the ITU standards.

We start this example by noting that there seems to be no uniform practice for writing these polynomials and hence their interpretation can result in resulting implementations between different bit construction tools.

So consider a system in which the output is arrived at by adding its 6th and 7th delay. We can then write this equation as :

$$y(n)=y(n-6)+y(n-7) \quad \text{(Eq. 6.4)}$$

or, in homogeneous form,

$$y(n)-y(n-6)-y(n-7)=0\,. \quad \text{(Eq. 6.5)}$$

This leads to the CE

$$1-z^{-6}-z^{-7}=0$$

The exponents of this equation are in descending order so that we can find the seven roots to this 7th degree equation using Matlab's companion matrix form to find roots through:

```
r=roots([1 0 0 0 0 0 -1 -1];
```

Then, in order to find the IC solution we formulate system of equations as a function of the power of the roots and a set of seven independent initial conditions to find the seven corresponding coefficients C_i as indicated in Eq 4.The initial conditions come knowing the first seven outputs of the scrambler impulse response which we find either by hand or by using Matlab's filter command.

The code is given by:

```
r=roots([1 0 0 0 0 0 -1 -1]);
rt=r.';
rt2=rt.^2;
rt3=rt.^3;rt4=rt.^4;rt5=rt.^5;rt6=rt.^6;
M=[1      1      1      1      1      1
1;rt;rt2;rt3;rt4;rt5;rt6];
IM=inv(M);
yimpulse=filter(1,[1  0  0  0  0  0  -1  -
1],[zeros(1,7)],[1 0 0 0 0 0 0]);
C=IM*yimpulse';
Ct=C.';
%   The   IC   equation   is   given   by
yIC67(n)=Ct*r.^(n-1);
% Find the first 127 coefficients using
the below for loop.
for n=1:127
    yIC67(n)=Ct*r.^(n-1);
end
```

The roots and the corresponding coefficents to be inserted into equation 4 are as follows

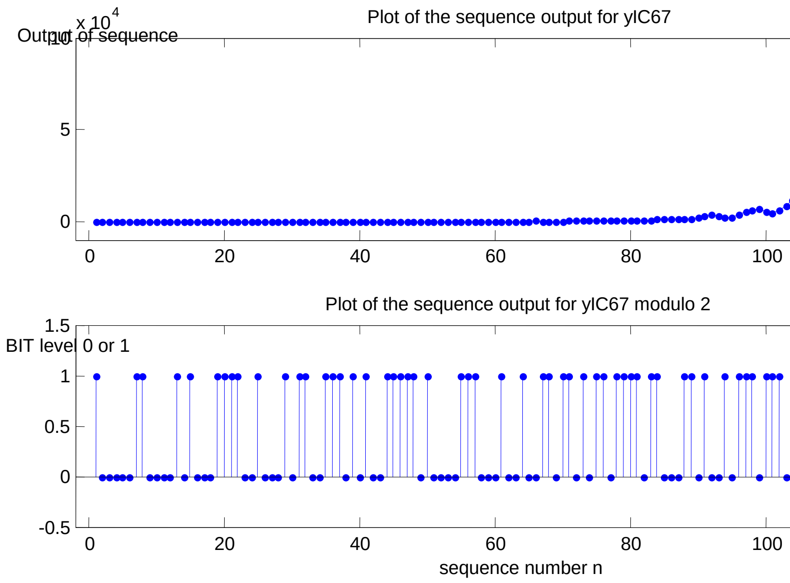
r1 – r7 : 1.11, 0.62 + 0.90i, 0.6271 - 0.90i, -0.36 + 0.95i, -0.36 - 0.95i, -0.81 + 0.26i, -0.81 - 0.26i

c1 – c7 : 0.15 - 0.00i, 0.15 - 0.01i, 0.15 + 0.01i, 0.15 - 0.02i, 0.15 + 0.02i, 0.11 - 0.0372i, 0.11 + 0.0372i.

Note that the magnitude of the roots are as follows:

Abs(r1-r7) : 1.11, 1.09, 1.09, 1.02, 1.02, 0.85, and 0.85.

Notice that some of the magnitudes are greater than 1. This indicates that the system is unstable. This means that as n increases, the sequence y(n) also increases. Now in our system of interest, that being scramblers, our resulting sequence should be recalculated using modulo 2 arithmetic, since we are in the bit domain. We were looking for a way to expose and exploit the underlying instability through these equations but have been unsuccessful. We have plotted the output sequence with and without the modulo 2 operation.



It is with the first plot where we would like to incorporate a measure of entropy in the context of Tsallis entropy. Tsallis entropy is a more general form of Shannon entropy. We know by looking at the magnitudes of the roots that the sequence will grow without bound. The rate of increase is in proportion to the magnitude of the root. Each scrambler has with a unique curve similar to the first plot.

The second plot shows how, under this implementation scheme which we have carefully annotated and defined, the sequence of bits resulting from an impulse response (a 1 follwed by 126 zeros) appears. This series of 127 bits is a deterministic sequence which, if we incorporated more zeros into the impulse ,would repeat itself every $2^7 - 1 = 127$ bits where 7 is the degree of the polynomial or number of delays in the Eq. 2. The understanding why there are 127 bits before the sequence repeats itself comes from the notion of a maximal length sequence associated with certain properties of Galois fields. The same properties og Galois fields are used to design EDAC's.

From the research on EDAC's we see that the coding process can be done in two domains: the systematic and non-systematic domains. In the former domain we take a message word and convolve it with a polynomial, which in form is exactly like that of a scrambler, to create a code word. In the latter form the message word is shifted to the left which then goes through a division process that 'systematically' rewrites the message word along with its remainder found in the division process. The convolution process is identified with the FIR filter while the division process is identified with the IIR filter. Although, in the theoretical sense, any FIR filter can be written as an IIR filter and visa versa, this is not exactly true when implementing the filters in a computer. A computer does well with the finiteness of a FIR filter but does not handle the element of infinity associated with IIR filters very well at all. But the comparison's are interesting. It turns out that associated with every scrambler/EDAC system, used in the context of modulo 2 arithmetic used to filter bits, carrying out the process can be done in either of the domains described above. As an observation we see that we can identify the short term application of the systematic EDAC division process with the long term IIR scrambling process and also identify the slightly longer term convolutional nonsystematic EDAC process with a shambling process. A subtle point to come out of the above plot is that although the DE if left free from the operation of modulo arithmetic grows without bound the sequence that results from applying modulo arithmetic results in a period sequence. This periodic sequence can in fact be used to scramble a bit stream as effectively as the

implemented system equation written in Eq. 4. The difference is that in Eq. 4 the output sequence is arrived at by using two taps (the 6th and 7th delay points) in the context of an IIR filter while in the context of a FIR or convolutional filter the output would be arrived at by substituting the 127 point sequence found from an impulse response to the scrambler, into the b_i's found on the right side of Eq.4 and setting the a_i coefficients to zero. The reason that the IIR filter is used over the FIR should be obvious; required are a lesser number of taps to implement in hardware.

Learning that we can in fact rewrite the IIR filter in terms of an FIR filter allows us to use a blind deconvolution algorithm to attempt to determine (blindly) the coefficients of the scrambler or likewise uncover the message word from a nonsystematic EDAC coding.scheme. Referring to a previous report we formulate a construction of a matrix system designed for a convolution or FIR process. The idea is that under certain assumptions, when made available are the outputs of two sequences of either codewords or scrambled bits that share a common underlying sequence, namely the scrambler or EDAC polynomial, the input sequences can be found and hence the scrambler or EDAC polynomial coefficients can be determined. Two problems have been identified with this approach. The first is that this approach works when the modulo 2 operation is not used. The second problem is that with scramblers the process used in the matrix method assumes an FIR filter but scramblers are clearly IIR. The 127 point sequence does not exactly replace the IIR sequence because to do so the sequence would have to be repeated infinitely many times. Hence, truncation is a problem. We will show what we have so far.

6.1 Entropy of Scrambled TypeA Data

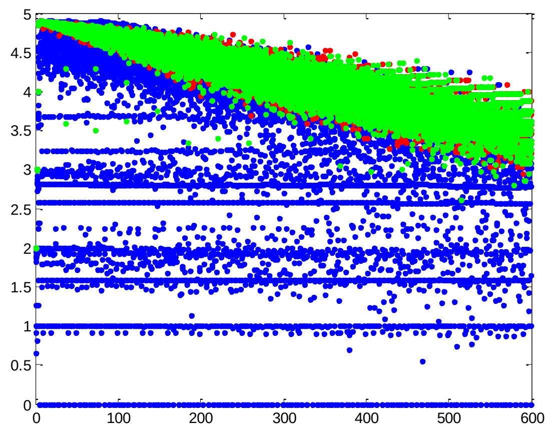
We investigate the entropy as a function of the window size for data scrambled with a number of different scramblers. The results are shown in Table 6.1. Note that for framed data, the entropy decays with the size of the window, because as the window grows larger, the regularity and predictability of the scrambler begins to manifest itself.

The outline of the code is:

```
M=[2,41;    2,4;    33,35;
    39,41;  31,35;
    31,39;  7,37;   6,8;
     2,3;    37,44;   29,35;
    35,39;    4,39;    6,45;
     6,41;    2,35; 6,42;43,45;
     4,43;    2,31;    1,7];
for k = 1:iterations
    [x,states] = filter(1,D,
        S((k1)*window+1:k*window),
        states);
    R = [ R, mod(x,2) ];
    states = mod(states,2);
end
R=R.^1.4;
for k3=1:length(cc)
    k2=cc(k3);
    pp=fix(length(R)/k2);
    v=reshape(R(1:pp*k2),k2,pp)';
    szv=size(v,1);
    b=2.^(0:k2-1);
    w=v*b';
    h=hist(w,nbins);
    h=h/sum(h);
    sh(k3,kk)=abs(
            sum(h.*log2(h+eps)));
end
```

The blue dots denote the impulse responses (lrs) of the scramblers, the red dots represent scrambled synthetic framed data and the green dots represent scrambled TypeA data.

Table 6.1. Entropy of Scrambled Data



6.2 Blind Descrambling

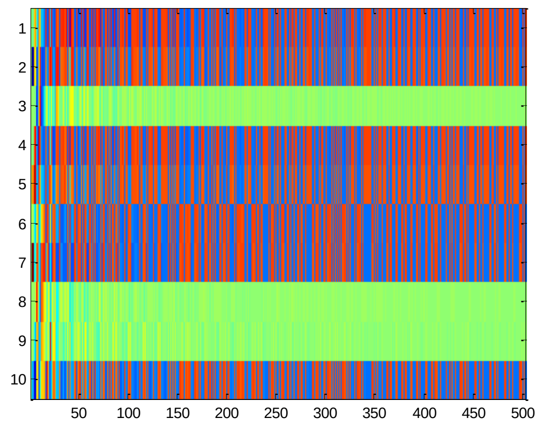
We have succeeded in extending the application range of the multipath mitigation algorithm to the area of blind descrambling, albeit only for the restricted case of short message lengths. This result is particularly interesting in view of the fact that in this setting, the multipath equations are not satisfied exactly, only approximately, due to truncation of polynomials. Therefore, strictly speaking, there is no precise mathematical justification for the algorithm to work. However, through numerical experiments, we have shown that there are cases, in particular, when the message length is short, where the algorithm succeeds.

We present such an example, the algorithm and the successful application of the algorithm and the corresponding solution in the following code snippet:

```
w=mod(filter(1,[1 0 0 -1 -1],
             zeros(1,200),
            [1 0 0 0]),2)
w=w(1:15);
r=repmat(w,1,200);
m1=[1 1 0 1 1];
m2=[1 1 0 0 1];
y1=(conv(r,m1));
y2=(conv(r,m2));
for k=2:504
   c1=convmtx(y1(1:k)',5);
   c2=convmtx(y2(1:k)',5);
   C=[c2 c1];
   CC=(C'*C);
   [aa ss]=eig(CC);
   aaak(:,k)=aa(:,1);
   sssk(k)=ss(1,1);
   ssskk(k)=ss(2,2);
   ssskkk(k)=ss(end,end);
end
figure(1)
subplot(3,1,1),plot(sssk)
subplot(3,1,2),plot(ssskk)
subplot(3,1,3),plot(ssskk-sssk)
figure(2)
imagesc(aaak)
```

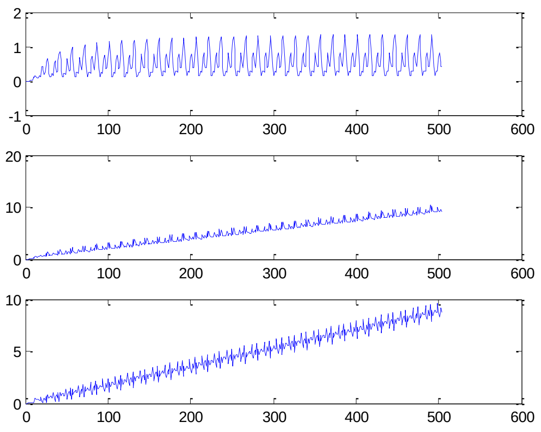
Table 6.2 shows that the algorithm succeeds in finding the correct solution as long as the size of the multipath matrix is chosen to be large enough. Note that each column in Fig 6.2 represents the solution and that solutions must first be normalized, because from the linear algebra point of view, any multiple of a solution is also a solution, while for practical purposes, the numbers are binary. In particular, note that the negative of a solution is also a solution, which explains the alternating red – blue pattern. Green appears in a band, because it represents zero.

Table 6.2. Blind descrambling



To provide some insight into the working of the multipath algorithm in this setting, we plot the smallest two eigenvalues and their difference as a function of the multipath matrix size in Table 6.3. Ideally, if the multipath equations were holding exactly, the smallest eigenvalue would be 0 and the corresponding eigenvector would be the solution. Note that the smallest eigenvalue is not zero, except for trivially small sizes of the multipath matrix. However, after the initial growth region, its value reaches a plateau. This is to be contrasted with the behaviour of the second smallest eigenvalue, which continues growing. In a sense, the smallest eigenvalue is "approximately zero", at least compared to the other eigenvalues. We therefore conjecture, that the eigenvector corresponding to the smallest eigenvalue continues to represent the solution, even though the theoretical justification is lacking, Our experiments show that this is indeed the case when the scrambled messages are very short. Unfortunately, it is not true in general.

Table 6.3. Roots Space



6.3 Fractal Structures in Root Space

During the investigation of the difference equations, their solutions and the relationship with the roots of characteristic polynomials, we have observed that the set of roots of characteristic polynomials exhibit interesting structures and in particular fractal behaviour. These results are presented in Fig. 6.4 and Fig. 6.5.

Table 6.4. Roots Space

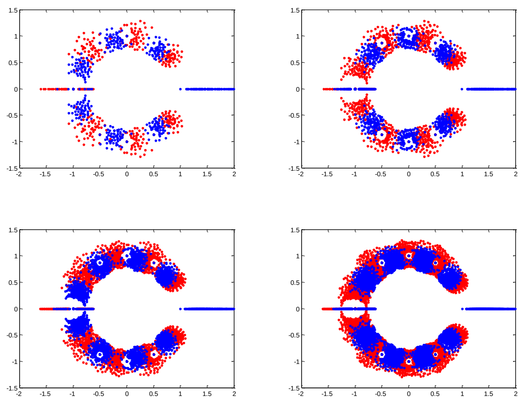
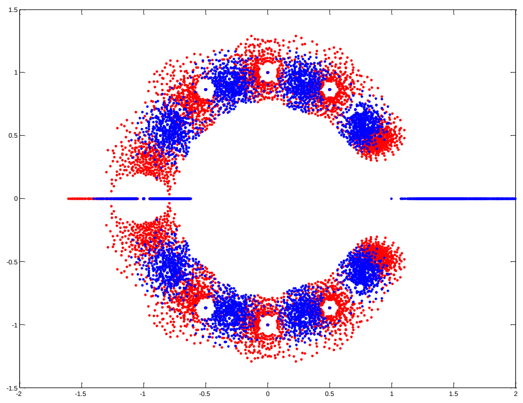


Table 6.5. Fractal Structures



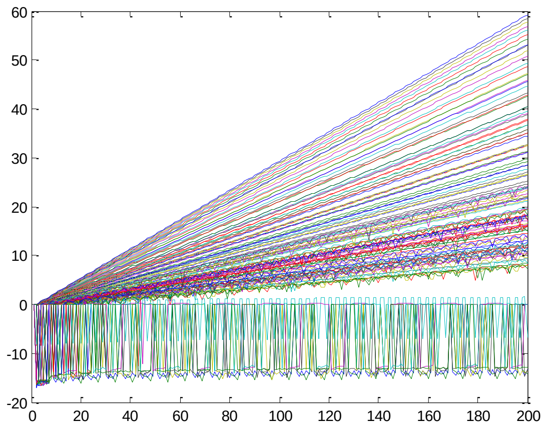
6.4 LRS Growth Rates

Fig. 6.6 summarizes our investigation of the growth rates of the linear recursive sequences defined by all the different binary polynomials of degree six. Each curve represents one such linear recursive sequence (before the modulo 2 operation). The vertical axis is logarithmic. Since all the curves have an asymptote, this shows that in the asymptotic region, the growth is exponential and it is determined by the largest root of the characteristic polynomial of the difference equation.

The brief sketch of the code is:

```
for k=1:2^n*2
  ss=s(k,:);
  r=roots(ss);
  rt=r.';
  rt2=rt.^2;
  rt3=rt.^3;
  rt4=rt.^4;
  rt5=rt.^5;
  rt6=rt.^6;
  m2=[ones(1,n+1);
      rt;rt2;rt3;rt4;rt5;rt6];
  Iaa=inv(m2);
  yss=filter(1,ss,[zeros(1,100)],
            [1 zeros(1,n)]);
  c=Iaa*yss(1:n+1)';
  ct=c.';
  for nn=1:200
     xss(nn,k)=ct*r.^(nn-1);
  end
end
plot(log10(abs(xss)))
```

Table 6.6. LRS Growth Rates



7. CONCLUSION AND OUTLOOK

Finite fields provide the mathematical foundation for understanding coding and scrambling. The first part of this research outlines the mathematical techniques which we believe will be help us in future research on blind decoding and descrambling.

In the second part, we present the results. We have succeeded in applying the multipath mitigation algorithm to blind descrambling for the case of short messages and observed fractal self similar structures in the root space of the scrambler, whose implications are as of yet unknown. The problem of bit sequences in the modulated domain is hard and deserves further study. We plan to address it in the future.

8. REFERENCES

[1] Todd K. Moon, Error Correction Coding. *Wiley*, 2005.

[2] Rudolf Lidl, Gunter Pilz, *Applied Abstract Algebra*. Springer, 1998.

[3] G. Birkhoff, T.C. Bartee, Modern Applied Algebra, McGraw-Hill, 1970.

[4] Salvatore Gravano, *Introduction to Error Correction Codes*, Oxford University Press, 2001.

[5] Berlekamp E. W., *Key Papers in the Development of Coding Theory*. IEEE Press, 1974.

[6] R.J. McEliece, Finite Fields for Computer Scientists and Engineers, Kluwer Academic Publishers, 1980.

[7] Philip Koopman, Tridib Chakravarty, Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks, *The International Conference on Dependable Systems and Networks, DSN-2004.*

[8] Parameter Sequencing using Shift Register Generators, National Technical ELINT Center, Informal Technical Note, 1-04, April 2004.

[9] Hamming(7,4) Code, Wikipedia, http://en.wikipedia.org/wiki/Hamming_7%2C4

[10] R. Gautier, G. Burel, J. Lettessier, O. Berder, *Blind Estimation of Scrambler Offset Using Encoder Redundancy*, 36 Asilomar Conference on Signals, Systems and Computers, Pacific Grove, California, USA, Nove 3-6 2002.

[11] Jacob H. Gunther, Lee Swindlehurst, *On the Use of Kernel Structure for Blind Equalization*, IEEE Transactions on Signal Processing, Vol 48, No 3, March 2000.