ERROR CONTROL CODES FROM ALGEBRA AND GEOMETRY – NOTES FOR SACNAS MINICOURSE

John B. Little
College of the Holy Cross

Edward Mosteig
Loyola Marymount University

September 25, 2004

Abstract. Communication of information often takes place over noisy channels. For reliability, it is often necessary to *encode* the transmitted information in such a way that errors can be detected and/or corrected when they occur. Designing schemes that achieve error control without introducing undue redundancy, and that admit efficient encoding and decoding, are the main goals of coding theory. Techniques from algebra and geometry have come to play an important role both in designing codes and in developing encoding and decoding algorithms. This minicourse will introduce the basic theory of error control codes, illustrate this with the example of the Reed-Solomon codes, then introduce an important recent idea – the construction of codes from order domains.

§1. Basics on Error Control Codes

Coding theory and information theory are relatively new subjects; their foundations were laid by Claude Shannon in a seminal paper [S] in 1948. Starting in the early 1950's many workers in communications and electrical engineering have developed coding schemes to ensure reliability of information transmission in areas such as

(1) communications with deep-space exploration craft,
(2) design of computer memory systems,
(3) the CD audio and DVD video systems,
(4) wireless telephony,

and many others.

As coding theory has grown, more and more tools from algebra have come to be useful for finding codes with good properties and for implementing encoding and decoding procedures. In this section, we will review some basic definitions. The references [HP], [McWS], and [vLi] are excellent sources for more detailed presentations of this basic material.

We will always consider a communications environment in which all messages are divided into "words" or blocks of a fixed length, k, formed using a finite alphabet with q symbols. The simplest case (also the one best adapted to electronic hardware) is an alphabet with two symbols, the binary digits $0, 1$. And indeed in most

applications, for instance in the codes used for the transfer of digital information within computer systems, and for storing information on compact disks, digital audio tape, or other media and retrieving it for use at a later time, q is either 2 or a power of 2. The alphabet with exactly two symbols can be identified with the finite field $\mathbb{F}_2$. Often, we consider all k-tuples as possible words that can appear in a message, so the collection of words may be identified with $\mathbb{F}_2^k$.

In order to detect and/or correct errors when they occur, some *redundancy* must be built into the information that is actually transmitted over the channel. One possible approach is to make the encoded form of a message consist of blocks or n-tuples of length $n > k$ over the same alphabet as that used for the message itself. Then the encoding and decoding operations can be described mathematically as functions:

$$E : \mathbb{F}_2^k \to \mathbb{F}_2^n$$

and

$$D : \mathbb{F}_2^n \to \mathbb{F}_2^k \cup \{ \text{ “error” } \},$$

where E is one-to-one, and $D \circ E$ is the identity mapping on $\mathbb{F}_2^k$. The function D will return the “error” value on some words in the complement of the image of E that cannot be decoded on the basis of the information known to the decoder. We call $C = Im(E)$ the set of *codewords*, or just the *code*, and the C obtained in this way are called *block codes* of length n over the alphabet $\mathbb{F}_2$.

When a codeword x is sent over the channel and a transmission error occurs, the effect is to replace the codeword x by a sum $x + e$ where $e \in \mathbb{F}_2^n$ is the *error vector* (this is the componentwise sum, using addition modulo 2). The word $y = x + e$ is received by the decoder, which then attempts to recover x itself. The number of nonzero entries in e determines how many of the entries in x have been corrupted. In order for the error to be detectable, $x + e$ must not be another codeword. Under many circumstances (for instance if the probability of an error occurring is the same in each location of each transmitted word), smaller numbers of nonzero entries in e are more likely than larger numbers. A good strategy is then to attempt to correct errors by finding the closest codeword to the received word (the “nearest neighbor” codeword) in the following sense.

Let $x, y \in \mathbb{F}_2^n$. The *Hamming distance* between x and y is defined to be

$$\begin{aligned} d(x, y) &= |\{i \in \{1, \ldots, n\} : x_i \neq y_i\}| \\ &= \text{number of nonzero entries in } x + y \bmod 2. \end{aligned}$$

For example, $d(11000111, 10100101) = 3$. Note that $d(x, 0)$, which is called the *weight* of x, is just the number of nonzero entries in x. It is an nice exercise to show that $d(x, y)$ satisfies all the properties of a *metric* or *distance function* on the finite set $\mathbb{F}_2^n$. In particular, we have a *triangle inequality*:

$$d(x, y) \leq d(x, z) + d(z, y)$$

for all $x, y, z \in \mathbb{F}_2^n$. If $x \in \mathbb{F}_2^n$ and $s \geq 0$, we write $B(x, s) = \{y \in \mathbb{F}_2^n : d(x, y) \leq s\}$ for the closed Hamming distance ball of radius s centered at x. Using this, we have the following fundamental statement about the error-detecting and error-correcting capability of codes.

(1.1) Proposition. *If a code $C \subset \mathbb{F}_2^n$ satisfies $d(x,y) \geq d$ for all distinct pairs $x, y \in C$, then any error vector of weight at most $d-1$ can be detected. Moreover, any error vector of weight at most $\lfloor (d-1)/2 \rfloor$ can be corrected by the "nearest-neighbor" decoding function:*

$$D(y) = \begin{cases} E^{-1}(x \in C : d(y,x) \text{ is minimal}) & \text{if } x \text{ is unique} \\ \text{"error"} & \text{if } x \text{ is not unique} \end{cases}$$

PROOF. First note that if $d(x,y) \geq d$ for all distinct pairs $x, y \in C$, then changing any $1 \leq r \leq d-1$ entries in a codeword never produces another codeword. Hence any nonzero error vector of weight at most $d-1$ produces a received word that can be distinguished from all codewords. So the fact than an error occurred can be *detected*, even though it may not be possible to determine from $y = x + e$ alone which codeword x was intended as part of the transmitted message.

For the second statement, we claim that if $d(x,x') \geq d \geq 2s+1$ for all distinct pairs of codewords x and x', then the balls $B(x,s)$ and $B(x',s)$ must be *disjoint*. If not, then for $y \in B(x,s) \cap B(x',s)$

$$d(x,x') \leq d(x,y) + d(y,x') \leq 2s$$

by the triangle inequality. But this contradicts our hypothesis. Hence if the error vector has weight $s = \lfloor (d-1)/2 \rfloor$ or less, the received word is in $B(x,s)$ but not in $B(x',s)$ for any $x' \neq x$, and nearest-neighbor decoding will correct the errors introduced in transmission. □

In real-world applications, the characteristics of the intended communications channel (in particular the probability that a symbol is transmitted incorrectly) play a major role in the choice of a code for a particular situation. Nearest-neighbor decoding can fail if the received word y from the channel contains too many errors, since then y may in fact be *farther from* the intended codeword x than it is from a different codeword x'. So when engineers compare codes, the most important parameters are

- $d = \min_{x \neq y \in C} d(x,y)$, the *minimum distance*, and
- $R = k/n$ (or more generally $\log_2(|C|)/n$ if $|C|$ is not a power of 2), called the *information rate*.

Good codes are ones for which $R = k/n$ is not too small (so the code is not extremely redundant), but for which d is also not too small. These conditions are clearly somewhat incompatible. While it is known by a famous result known as Shannon's Theorem (see [vLi]) that there exist codes with information rate nearly 1 for which the probability of decoding a random received word incorrectly using the nearest-neighbor function is arbitrarily small, it may be necessary to take n very large (hence large k as well) to achieve this, increasing the amount of work (time, energy, etc.) needed to encode or decode messages. Hence identifying good codes is a delicate balancing act, and much effort has been devoted both to finding explicit good codes, and to developing theoretical bounds on the parameters of codes. For future reference we mention two of the simplest of these bounds.

(1.2) Proposition. *Fix n, d and let $b = b(n,d)$ equal the number of n-tuples in the ball $B(x, d-1)$ centered at an arbitrary $x \in \mathbb{F}_2^n$:*

$$b = \sum_{m=0}^{d-1} \binom{n}{m}.$$

Let $A(n,d)$ be the largest number of codewords possible for a code C in $\mathbb{F}_2^n$ with minimum distance d.

1) (Singleton bound)

$$A(n,d) \leq 2^{n-d+1}.$$

Hence for any code $C \subset \mathbb{F}_2^n$ with 2^k codewords and minimum distance d, $k \leq n-d+1$.

2) (Gilbert-Varshamov bound)

$$A(n,d) \geq 2^n/b.$$

The proof of the Singleton Bound is obtained by deleting any fixed set of $d-1$ entries from all the codewords of C. The Gilbert-Varshamov bound follows from the fact that for a code with the maximum number of codewords for the given n, d, there cannot be any elements of $\mathbb{F}_2^n$ outside the union $\cup_{x \in C} B(x, d-1)$. (If there were, we could add such a word to the code and maintain the minimum distance.)

While any collection C of codewords in $\mathbb{F}_2^n$ can be considered as a code, we will restrict our attention from now on to codes with some *additional algebraic structure*. Namely, we will consider only *linear* block codes, for which the set C is a k-dimensional vector subspace of $\mathbb{F}_2^n$. That is, for a linear code, the set of codewords is closed under vector sums (modulo 2, as always). The closure under scalar multiplication required for the vector space structure is automatic in this case, because the only scalars are $0, 1 \in \mathbb{F}_2$.

One reason for making this restriction is that, because linear algebra provides powerful tools, linear codes with reasonable parameters are simply easier to construct explicitly than comparably good arbitrary codes. The construction of the Reed-Solomon codes in the next section is a perfect example of this. In addition, linear codes admit both

- encoding algorithms that require much less stored information, and
- decoding algorithms that are much more efficient

than the methods that are available for arbitrary codes.

For instance, to compute the encoding function for a completely general code where the codewords had no extra symmetries, it would be necessary to store all 2^k of the codewords and do some form of table look-up to find the codeword corresponding to each word $x \in \mathbb{F}_2^k$ in the message. While this look-up can be done efficiently, it requires an impractically large amount of storage for codes with realistic n and k for some applications. (For instance, one of the codes used in the CD digital audio system can be viewed as a code over $\mathbb{F}_2$ with $n = 256$ and $k = 224$. Hence there are 2^{224} different codewords, each of which is a string of 256 elements of $\mathbb{F}_2$–a total of over 6.4×10^{60} gigabytes of information!)

On the other hand, a linear code C can be completely specified by any basis—k vectors instead of 2^k of them. Moreover, the encoding operation can be performed via matrix multiplication. It is customary to write the $x \in \mathbb{F}_2^k$ and the codewords in $\mathbb{F}_2^n$ as row vectors. For any $k \times n$ matrix G whose rows form a basis for C, the formula $E(x) = xG$ defines an encoding function $E : \mathbb{F}_2^k \to \mathbb{F}_2^n$ for C. The matrix G is called a *generator matrix* for C.

Example 1. In $\mathbb{F}_2^7$ consider the code C given by the generator matrix

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

For this code, we have $n = 7, k = 4, d = 3$. Since $\lfloor (3-1)/2 \rfloor = 1$, this code can correct any single bit error in word of length 7. How we can tell what d is? For linear codes, if $x, y \in C$, then $x - y \in C$ too. Hence

$$\min_{x \neq y \in C} d(x, y) = \min_{x \neq y \in C} d(x - y, 0) = \min_{z \neq 0 \in C} d(z, 0)$$

In other words: For linear codes, the minimum distance is the same as the minimum "weight" of the nonzero codewords. It can be checked that $d = 3$ here by explicitly writing out all 15 nonzero codewords and noting their weights. Note that if the encoding function $E(x) = xG$ is used, then the four entries of x are copied into the first four entries of $E(x)$.

The code given by G above has another interesting property: every word in $\mathbb{F}_2^7$ is either a codeword, or Hamming distance 1 from a unique codeword (that is, the union of the Hamming balls of radius 1 centered at the codewords is all of $\mathbb{F}_2^7$). In other words, the Hamming balls centered at the codewords are "packed together" in an extremely efficient way – there is no left-over space at all. There is a infinite family of such codes containing this one (parameters $n = 2^r - 1$, $k = 2^r - r - 1$, $d = 3$ for all $r \geq 1$) that all have this property. They are called the *Hamming codes* (see [HP] or [McWS]).

A rudimentary decoder for this code would use a "check matrix" H for C such as

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

for which the codewords $x \in C$ (written as rows) are the solutions of $xH = 0$. Assuming the received word contains no more than 1 bit error,

(1) Given a received word x' compute $s = x'H$; if $s = 0$, then x' is a codeword itself; if not, then
(2) Toggle each bit in x' in turn to get words y, until a solution of $yH = 0$ is found.

Even better methods are known too if we are allowed to store some precomputed information about the code, for instance the "syndrome decoding" method (see [HP] or [McWS]).

For future reference, generalizing the observation above, we see that if a linear code is given by a generator matrix of the form $G = (I|B)$ where I is a $k \times k$ identity matrix, and B is a $k \times (n-k)$ block, then the first k entries in $E(x) = xG$ will be the entries from x itself. Encoding functions for which all the symbols of the

input word appear unchanged in some components of the corresponding codeword are said to be *systematic encoders*. It is customary to call those components of the codewords the *information positions*. The remaining components of the codewords are called *parity checks*. This name comes from the fact that one simple error-detection scheme for binary codes is to require that all codewords have an even (or odd) number of nonzero digits. To ensure this, one could simply append another 0 or 1 to each message word to adjust the parity of the number of 1's. If one bit error (in fact, any odd number of errors) is introduced in transmission, that fact can be recognized by counting the number of 1's in the received word. A similar scheme, in which an extra check digit is added to credit card numbers, is used to detect transmission errors when your credit is checked during purchases.

Systematic encoders are sometimes desirable from a practical point of view because the information positions can be copied directly from the word to be encoded; only the parity checks need to be computed. There are corresponding savings in the decoding operation as well. If information is systematically encoded and it is determined that no errors occur in transmission, the words in the message can be obtained directly from the received words by simply removing the parity checks. It is perhaps worthwhile to emphasize again at this point that the goal of the coding schemes we are considering here is primarily *reliability* of information transmission, not secrecy!

§2. Reed-Solomon Codes

In this section we will study the construction of a class of codes having very good properties. These codes were first constructed in 1960 by Irving Reed and Gustave Solomon, so they are known as Reed-Solomon codes. Our presentation will be quite similar to the original way these codes were constructed.

To describe the Reed-Solomon codes, we need to introduce explicit fields larger than $\mathbb{F}_2 = \{0, 1\}$ to be used as code alphabets. Sticking to the binary setting, we might take the set of *strings* of $0, 1$'s of a fixed length r as the alphabet. For instance, strings of length $r = 4$ would give 2^4 distinct symbols:

$$0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,$$
$$1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.$$

In order to work with the set-up of linear codes, though, this set must be given the structure of *a field*.

As a quick review, we now mention a few definitions. A set R with two binary operations, '$+$' (addition) and '$\cdot$' (multiplication), is called a *commutative ring* if there exist two distinct elements $0, 1 \in R$ such that for all $a, b, c \in R$

- (Commutativity) $a + b = b + a, \quad a \cdot b = b \cdot a$
- (Associativity) $(a + b) + c = a + (b + c), \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- (Identity) $a + 0 = a, \quad a \cdot 1 = a$
- (Additive Inverses) $\forall a \in R$, $\exists (-a) \in R$ such that $a + (-a) = 0$
- (Distributivity) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

If R has the additional property that each nonzero element has a multiplicative inverse, then R is called a field. Common examples of fields are $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$, though we will focus on finite fields.

We now describe how to make strings of binary symbols into a field. We can interpret a string $\beta_3\beta_2\beta_1\beta_0$ ($\beta_i \in \mathbb{F}_2$) as a polynomial in a new variable α:

$$\beta_3\alpha^3 + \beta_2\alpha^2 + \beta_1\alpha + \beta_0 \tag{1}$$

The set of all 16 such expressions will be denoted by $\mathbb{F}_{16}$, though we need to do some work to show that this is actually has the structure of a field.

The addition operation presents no problems – we will use the usual polynomial addition (same as vector addition in $\mathbb{F}_2^4$). We also need a *multiplication operation* and the key point is that we know how to multiply polynomials. For example,

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

But of course the degree of the product is *too large* here. To reduce to the proper range of degrees ≤ 3, we can *divide* by some polynomial of degree 4 in α (using "polynomial long division") and take the *remainder* of the product (recall that the remainder is either 0 or of degree strictly less than the degree of the divisor, because the division process continues until the leading term of the divisor is larger than the degree of the remaining intermediate dividend).

This much works for any divisor polynomial $h(\alpha)$ of degree 4. But, do we always get a field this way? The answer is *no*. For instance, if $h(\alpha) = \alpha^4 + 1$, then $(\alpha^2 + 1)(\alpha^2 + 1) = \alpha^4 + 1$, so when we divide, the remainder will work out to be *zero*! On the other hand a field cannot have nonzero elements a, b such that $ab = 0$. So we must take an *irreducible* polynomial $h(\alpha)$ as our divisor – one which does not factor into a product of polynomials of positive degree. (These are analogous to the prime numbers in the integers.)

For instance, it can be checked that $h(\alpha) = \alpha^4 + \alpha + 1$ is irreducible. Once again, multiplication is polynomial multiplication of the expressions (1), followed by reduction using the basic relation $\alpha^4 + \alpha + 1 = 0$. For instance, on our example above, this yields

$$\begin{aligned}(\alpha^3 + 1)(\alpha^2 + 1) &= \alpha^5 + \alpha^3 + \alpha^2 + 1 \\ &= \alpha(\alpha^4 + \alpha + 1) + \alpha^3 + \alpha + 1 \text{ by division}\end{aligned}$$

(recall – the coefficients are integers mod 2!) Hence, in $\mathbb{F}_{16}$ (constructed using $h(\alpha) = \alpha^4 + \alpha + 1$), we have

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

One easy way to see that this method satisfies properties needed to see we have a field is to check that the powers $1, \alpha, \alpha^2, \ldots, \alpha^{14}$ are all distinct, and $\alpha^{15} = 1$. Hence

- The powers of α give all the nonzero elements of $\mathbb{F}_{16}$, and
- Each element α^k has a multiplicative inverse α^{15-k}.

We call α a *primitive element* for $\mathbb{F}_{16}$.

The following theorem shows that we can construct finite fields of all sizes p^r where p is a prime number, and $r \geq 1$. These are sometimes called *Galois fields* after the 19th century French mathematician Évariste Galois, who discovered them.

(2.1) Theorem. *Let p be prime in $\mathbb{Z}$.*

(1) *The set of integers mod p is a field denoted $\mathbb{F}_p$.*
(2) *For all p and all $r \geq 1$, there are irreducible polynomials of degree r with coefficients in $\mathbb{F}_p$.*
(3) *Let $h(\alpha)$ be an irreducible polynomial of degree r with coefficients in $\mathbb{F}_p$. The set of polynomials of degree $\leq r-1$ in α with coefficients in $\mathbb{F}_p$, under the usual addition and multiplication defined by $f(\alpha) \cdot g(\alpha) = \overline{f(\alpha)g(\alpha)}^h$, where $\overline{F}^h$ denotes the remainder on division by h, is a field of size p^r denoted $\mathbb{F}_p[\alpha]/\langle h(\alpha)\rangle$.*
(4) *Different choices of irreducible h of the same degree yield isomorphic fields, which are all denoted $\mathbb{F}_{p^r}$.*
(5) *Every field $\mathbb{F}_{p^r}$ has a primitive element (an element β whose powers yield all nonzero elements of $\mathbb{F}_{p^r}$).*

Proofs may be found in [HP], [P], or standard texts on abstract algebra. Extensive tables of irreducible polynomials in $\mathbb{F}_p[x]$ have been compiled to provide constructions of these fields for use in coding theory and other areas. See [McWS], for instance.

We will use this fact in the following way. Suppose we want to construct codes attaining the Singleton bound over a fixed finite field $A = \mathbb{F}_q$, where $q = p^r$ for some $r \geq 1$. (These are called MDS, or "maximum distance separable," codes). Restricting to codes of length $n = q$, there is a way to do this using a basic fact about polynomials over a field. Fix an integer $k \leq q$. Polynomials of degree $< k$ have at most $k-1$ roots in $\mathbb{F}_q$, and some have precisely that many roots.

We can use that observation and write L_k for the set of all polynomials in $\mathbb{F}_q[x]$ of degree $< k$. We will always assume $k < q$ here. L_k is a vector space over $\mathbb{F}_q$ of dimension k. For each polynomial $f \in L_k$, we construct a word in $\mathbb{F}_q^q$ by *evaluating* f at the elements of $\mathbb{F}_q$, to get a q-tuple (letting α be a primitive element),

$$(f(0), f(1), f(\alpha), \ldots, f(\alpha^{q-2}))$$

(recall $\alpha^{q-1} = 1$). When we do this, we get a word with

(1) at most $k-1$ zero entries, hence
(2) at least $q-(k-1) = q-k+1 = n-k+1$ nonzero entries (and some have exactly $k-1$ zero entries).

The set of *all* such words is a linear code since L_k is a vector space, and the evaluation mapping is linear. Hence the resulting code will have *minimum distance* $d = n-k+1$ (if k is small relative to q).

Using all q elements $0, 1, \alpha, \ldots, \alpha^{q-2}$ of $\mathbb{F}_q$, we get the so-called *extended Reed-Solomon codes*. The Reed-Solomon codes themselves come from evaluating only at the nonzero elements of the field (omitting the $f(0)$ entry to get a word in $\mathbb{F}_q^{q-1}$).

Summarizing, we have the following result.

(2.2) Theorem. *We write $\mathbb{F}_q[x]$ for the ring of all polynomials in x with coefficients in $\mathbb{F}_q$. Pick a primitive element α for $\mathbb{F}_q$, and write the nonzero elements of $\mathbb{F}_q$ as*

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

Let $k < q$ and $L_k = \{f \in \mathbb{F}_q[x] : \deg f < k\}$. Write

$$\begin{aligned} ev : L_k &\to \mathbb{F}_q^{q-1} \\ f &\mapsto (f(1), f(\alpha), \ldots, f(\alpha^{q-2})). \end{aligned}$$

Then $Im(ev)$ is a linear code with $n = q-1$, dimension k, and minimum distance $d = n-k+1 = q-k$, called a Reed-Solomon code, $RS(k,q)$. All Reed-Solomon codes reach the Singleton bound by construction – they are MDS codes: $k = n-d+1$, or $d = n-k+1$.

For example, using the standard monomial basis

$$\{1, x, x^2, x^3, \ldots, x^{k-1}\}$$

for L_k, the Reed-Solomon code $RS(3,16)$ (parameters: $n = 15, k = 3, d = 13$ over $\mathbb{F}_{16}$, so $16^3 = 4096$ distinct codewords) has generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^7 & \alpha^8 & \cdots & \alpha^{14} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{14} & \alpha & \cdots & \alpha^{13} \end{pmatrix}.$$

Reed-Solomon codes are probably the most commonly used codes in certain situations where errors tend to occur in "bursts" rather than randomly. This includes communication to and from deep-space exploration craft, the CD digital audio system, and many other applications. Reed-Solomon codes are an important component of the coding techniques used there (but also not the whole story—see the papers [McES] and [Imm] for instance). There are several reasons for this. The first reason is that Reed-Solomon and other block codes can correct relatively long bursts of errors on the bit level, even if the minimum distance d is relatively small. To see the idea, note that the entries of a codeword for a block code over a field $\mathbb{F}_{2^r}$ may be represented as strings of r bits: If $\mathbb{F}_{2^r} = \mathbb{F}_2[\alpha]/\langle h(\alpha)\rangle$, where h irreducible of degree r, then an element $\beta_{r-1}\alpha^{r-1} + \cdots + \beta_1\alpha + \beta_0$ of $\mathbb{F}_{2^r}$ can be identified with the vector $(\beta_{r-1}, \ldots, \beta_1, \beta_0) \in \mathbb{F}_2^r$. Under this identification, a Reed-Solomon codeword is represented by a string of $(2^r - 1)r$ bits. A burst of $2r$ consecutive bit errors, for instance, will change at most three of the entries of the codeword, when they are viewed as elements of $\mathbb{F}_{2^r}$. Hence by Proposition (1.1) if $d \geq 7$, for instance, then any consecutive burst of $2r$ bit errors can be corrected. On the other hand, if the errors are located in $2r$ arbitrary entries in the word, they may not be correctable with a code of that minimum distance.

The last, and most important, reason that Reed-Solomon codes are attractive is that they have additional algebraic structure that greatly facilitates the encoding and decoding operations. To see the idea, consider the generator matrix G for the Reed-Solomon code $RS(k,q)$ constructed by evaluating the monomials $\{1, x, x^2, \ldots, x^{k-1}\}$ at the $\alpha^\ell \in \mathbb{F}_q \setminus \{0\}$. The ith row of G has the form

$$((1)^{i-1}, (\alpha)^{i-1}, (\alpha^2)^{i-1}, \ldots, (\alpha^{q-2})^{i-1}).$$

Cyclically permuting this row, we obtain

$$((\alpha^{q-2})^{i-1}, (1)^{i-1}, (\alpha)^{i-1}, \ldots, (\alpha^{q-3})^{i-1}),$$

which is equal to

$$\alpha^{(i-1)(q-2)} \cdot ((1)^{i-1}, (\alpha)^{i-1}, (\alpha^2)^{i-1}, \dots, (\alpha^{q-2})^{i-1})$$

because $\alpha^{q-1} = 1$. Thus, a cyclic permutation of the ith row yields a scalar multiple of the same row—it is also one of the Reed-Solomon codewords! The cyclic permutation is a linear mapping S on $\mathbb{F}_q^{q-1}$, and we have just seen that there is a basis of $RS(k,q)$ consisting of *eigenvectors* for S. It follows that the Reed-Solomon code $RS(k,q)$ is *invariant* under S, since all the codewords are linear combinations of the rows of G.

Linear codes $C \subset \mathbb{F}_q^n$ that are invariant under the cyclic permutation

$$\begin{aligned} S : \mathbb{F}_q^n &\to \mathbb{F}_q^n \\ (x_1, x_2 \dots, x_n) &\mapsto (x_n, x_1, \dots, x_{n-1}) \end{aligned}$$

are called *cyclic codes*. The observations above give the proof of the following fact.

(2.3) Proposition. *For all q and all $k < q$, the Reed-Solomon code $RS(k,q)$ is cyclic.*

To understand the full meaning of this observation, we need to make one further identification. Namely, given a codeword $c \in RS(k,q)$:

$$c = (c_0, c_1, \dots, c_{q-2}) = (f(1), f(\alpha), \dots, f(\alpha^{q-2})),$$

where $f \in L_k \subset \mathbb{F}_q[x]$, we can use the entries in c as the coefficients in another polynomial in a new variable t. We write $\psi(c)$ for this polynomial:

$$\psi(c) = c_0 + c_1 t + \cdots + c_{q-2} t^{q-2}. \tag{2}$$

and call it the "polynomial form" of the Reed-Solomon codeword c.

It follows from the algebra of this situation that the polynomial forms $\psi(c)$ of the codewords $c \in RS(k,q)$ are all *divisible* by a polynomial called the *generator polynomial* for the Reed-Solomon code. (This follows from facts about the algebra of polynomials developed in the next section. Unfortunately, we do not have the time to go into this in detail; see [HP] or [P] for full details.)

To identify the generator polynomial for $C = RS(k,q)$, we may proceed as follows. From (2), every element of $\psi(C)$ has the form

$$\psi(c) = c_0 + c_1 t + \cdots + c_{q-2} t^{q-2}$$

where we obtain the coefficients

$$c_i = \sum_{j=0}^{k-1} a_j (\alpha^i)^j$$

by evaluating some fixed $f(x) = \sum_{j=0}^{k-1} a_j x^j$ at $x = \alpha^i$ for $i = 0, \dots, q-2$. Substituting these expressions for c_i into $\psi(c)$ and interchanging the order of summation we see:

$$\begin{aligned} \psi(c) &= \sum_{i=0}^{q-2} \left(\sum_{j=0}^{k-1} a_j (\alpha^i)^j \right) t^i \\ &= \sum_{j=0}^{k-1} a_j \left(\sum_{i=0}^{q-2} (\alpha^j t)^i \right). \end{aligned} \tag{3}$$

In $\mathbb{F}_q$, the roots of $0 = 1 + z + z^2 + \cdots + z^{q-2}$ are precisely the $z \neq 0, 1$. Hence the inner sum in the last line of (3) is equal to zero provided that $\alpha^j t \neq 0, 1$. The whole sum equals zero if $\alpha^j t \neq 0, 1$ for all $j = 0, \ldots, k-1$, or equivalently if $t \in \{\alpha, \alpha^2, \ldots, \alpha^{q-k-1}\}$. Since $c \in RS(k, q)$ was arbitrary, this computation shows that for all c, $\psi(c)$ from (2) (viewed as a polynomial in $\mathbb{F}_q[t]$) has $t = \alpha, \alpha^2, \ldots, \alpha^{q-k-1}$ as roots. Consequently, every $\psi(c)$ is divisible by

$$g(t) = (t - \alpha)(t - \alpha^2) \cdots (t - \alpha^{q-k-1}). \tag{4}$$

In fact, we have:

(2.4) Proposition. *The polynomial $g(t)$ from (4) is the generator polynomial for $RS(k, q)$.*

Since the minimum distance of a Reed-Solomon code satisfies $d = q - k$, the generator polynomial can also be written as

$$g(t) = (t - \alpha)(t - \alpha^2) \cdots (t - \alpha^{d-1}).$$

For example, the Reed-Solomon code $RS(3, 16)$ from above has

$$g(t) = (t - \alpha)(t - \alpha^2)(t - \alpha^3) \cdots (t - \alpha^{12})$$

since $d = 15 - 3 + 1 = 13$.

§3. The Euclidean Algorithm for Polynomials

In this section, we will introduce another key aspect of the algebra of polynomials in one variable over a field, the Euclidean Algorithm. This process for computing a greatest common divisor goes back to the ancient Greeks, and was used first in the case of ordinary integers. The ring of integers $\mathbb{Z}$ and the polynomial rings $\mathbb{F}_q[x]$ have many of the same properties, because both have well-behaved *division algorithms*. We have already used polynomial division in constructing our finite fields. Here is a precise statement of the characterization of the quotient and remainder.

(3.1) Theorem. *(Division algorithm in $\mathbb{F}_q[x]$) Let $f(x), g(x)$ be nonzero polynomials in $\mathbb{F}_q[x]$. Then there exist unique polynomials $q(x)$ and $r(x)$ satisfying*

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

where either $r(x)$ is the zero polynomial, or else $\deg(r(x)) < \deg(g(x))$.

(Note: the degree of a polynomial is by definition the largest power of the variable appearing with a nonzero coefficient in the polynomial. The degree is not defined for the zero polynomial, and that is why the theorem is stated in the way given here.)

The first consequence of this division algorithm concerns special subsets of the polynomial ring called *ideals*. An ideal $I \subset \mathbb{F}_q[x]$ is a nonempty subset that is closed under sums, and also closed under multiplication by all polynomials. That is, if $f, g \in I$, then $f + g \in I$. Moreover, if $f \in I$ and h is any polynomial, then $hf \in I$.

For example, let $\beta \in \mathbb{F}_q$. Then the set I of all polynomials $f \in \mathbb{F}_q[x]$ satisfying $f(\beta) = 0$ is an ideal in $\mathbb{F}_q[x]$. It is easy to check that both defining properties are satisfied here: If $f(\beta) = 0$ and $g(\beta) = 0$, then $(f + g)(\beta) = f(\beta) + g(\beta) = 0$. Moreover, if h is any polynomial $(hf)(\beta) = h(\beta)f(\beta) = h(\beta) \cdot 0 = 0$.

Theorem (3.1) implies the following statement about ideals.

(3.2) Theorem. *Every ideal I in $\mathbb{F}_q[x]$ is generated by a single polynomial $g(x)$ in the sense that*

$$I = \{q(x)g(x) : q(x) \in \mathbb{F}_q[x]\}.$$

Before proceeding to the proof, we note that ideals of the form described here – ideals generated by a single element of the ring – are called *principal ideals*. So this Theorem says that *every* ideal in the polynomial ring $\mathbb{F}_q[x]$ is principal. There is also a corresponding statement in the ring of integers. Moreover, it is this fact that underlies the existence of the generator polynomial for Reed-Solomon and other cyclic codes as in Proposition (2.4).

PROOF. If $I = \{0\}$, then the zero polynomial acts as the generator $g(x)$. So from now on, we assume that I contains nonzero polynomials. Let $g(x)$ be the nonzero monic (leading coefficient 1) polynomial of minimal degree in I. It is easy to see that $g(x)$ is unique, since if there were two monic polynomials of minimum degree in I, then their difference would be in I, but of smaller degree. Now let $f(x)$ be any other polynomial in I, and apply the division algorithm (3.1). We get $f(x) = q(x)g(x)+r(x)$ where either $r(x)$ is the zero polynomial, or else $\deg(r(x)) < \deg(g(x))$. But this equation implies that $r(x) = f(x) - q(x)g(x)$. So by the properties of an ideal, $r(x) \in I$. The polynomial $r(x)$ cannot have smaller degree than $g(x)$, since we chose $g(x)$ to have minimal degree among the nonzero elements of I. Therefore, $r(x) = 0$, which shows that $f(x) = q(x)g(x)$. Hence I is contained in the set of multiples of $g(x)$. But the other inclusion is automatic by the definition of an ideal. Hence we have the equality of sets claimed in the statement of the theorem. □

Another way to make an ideal in $\mathbb{F}_q[x]$ is to consider all "linear combinations" of any two given polynomials with polynomial coefficents:

$$\langle f(x), g(x)\rangle = \{a(x)f(x) + b(x)g(x) : a(x), b(x) \in \mathbb{F}_q[x]\}.$$

Theorem (3.2) says that these ideals are principal ideals too, and the generator polynomial $d(x)$ for this ideal $\langle f(x), g(x)\rangle$ is called the *greatest common divisor*, or GCD, of $f(x)$ and $g(x)$. The Euclidean Algorithm is a method for computing this greatest common divisor, and an extension provides the polynomials $a(x), b(x)$ too.

The Euclidean Algorithm works as follows to compute the GCD of two polynomials $f(x)$ and $g(x)$. We begin by assuming $\deg(f(x)) \geq \deg(g(x))$ and we divide $g(x)$ into $f(x)$ using (3.1). Then we divide the remainder from the first division into $g(x)$, the remainder from the second division into the first remainder, and so on, until a zero remainder occurs. (This must happen eventually, since the degree of the remainder decreases at each step.) In symbols, we can write the computations as follows:

$$\begin{aligned}
f(x) &= q_0(x)g(x) + r_1(x)\\
g(x) &= q_1(x)r_1(x) + r_2(x)\\
r_1(x) &= q_2(x)r_2(x) + r_3(x)\\
&\vdots\\
r_{k-1}(x) &= q_k(x)r_k(x) + 0
\end{aligned}$$

Then the *last nonzero remainder* is the GCD: $d(x) = r_k(x)$.

Here is an example. Consider the polynomials $f(x) = x^6 + x^5 + x^3 + x^2$ and $g(x) = x^6 + x^4 + x + 1$ in $\mathbb{F}_2[x]$. Carrying out the process above, we find:

$$\begin{aligned} f(x) &= 1 \cdot g(x) + x^5 + x^4 + x^3 + x^2 + x + 1 \\ g(x) &= (x+1)(x^5 + x^4 + x^3 + x^2 + x + 1) + x^4 + x \\ x^5 + x^4 + x^3 + x^2 + x + 1 &= (x+1)(x^4 + x) + x^3 + 1 \\ x^4 + x &= x(x^3 + 1) + 0 \end{aligned}$$

The sequence of remainders is

$$\begin{aligned} r_1(x) &= x^5 + x^4 + x^3 + x^2 + x + 1 \\ r_2(x) &= x^4 + x \\ r_3(x) &= x^3 + 1 \\ r_4(x) &= 0 \end{aligned}$$

So the last nonzero remainder $r_3(x) = x^3 + 1$ is the GCD $d(x)$ of $f(x)$ and $g(x)$. (In fact $f(x) = (x^3 + x^2)d(x)$ and $g(x) = (x^3 + x + 1)d(x)$, so $d(x)$ is a common divisor of $f(x)$ and $g(x)$).

In the next section, we will see that a very good *decoding algorithm* for Reed-Solomon codes is based on the kind of computations done here, and in the following "extended" version of the Euclidean Algorithm that computes $d(x)$ together with the $a(x), b(x)$ that give $d(x) = a(x)f(x) + b(x)g(x)$. We first introduce the notation $f(x) = r_{-1}(x)$ and $g(x) = r_0(x)$ to give a uniform form for the steps in the successive divisions. So every line of the above description can be written as

$$r_{k-1}(x) = q_k(x)r_k(x) + r_{k+1}(x)$$

for $k = 0, 1, 2, \ldots$. Then the statement of the Extended Euclidean Algorithm is as follows.

(3.3) Algorithm.

Input: *nonzero* $f(x), g(x)$
Output: $d(x), a(x), b(x)$

$r_{-1} := f; r_0 := g$
$a_{-1} := 1; a_0 := 0$
$b_{-1} := 0; b_1 := 1$
$k := 0$
WHILE $r_k \neq 0$ *DO*
  divide r_k *into* r_{k-1} : $r_{k-1} = q_k r_k + r_{k+1}$
  $a_{k+1} := a_{k-1} - q_k a_k$
  $b_{k+1} := b_{k-1} - q_k b_k$
  $k := k + 1$

Note: The polynomials $a(x), b(x)$, and $d(x)$ are the final values $a_k(x)$, $b_k(x)$, and $r_k(x)$, respectively.

We will present a nice tabular format for organizing and carrying out these calculations in the minicourse. This format is shown in the following example.

With the polynomials $f(x) = x^6 + x^5 + x^3 + x^2$ and $g(x) = x^6 + x^4 + x + 1$ as in our previous example, we get the following results:

k	r_k	q_k	a_k	b_k
-1	$x^6 + x^5 + x^3 + x^2$		1	0
0	$x^6 + x^4 + x + 1$	1	0	1
1	$x^5 + x^4 + x^3 + x^2 + x + 1$	$x + 1$	1	1
2	$x^4 + x$	$x + 1$	$x + 1$	x
3	$x^3 + 1$	x	x^2	$x^2 + x + 1$

We have

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

in $\mathbb{F}_2[x]$ as claimed.

Complete proofs for the Extended Euclidean Algorithm can be found in many abstract and computational algebra texts.

§4. Reed-Solomon Decoding

We now turn to the decoding problem for Reed-Solomon codes. Several different but related extremely efficient decoding algorithms for Reed-Solomon and related codes have been developed. And indeed, the fact that they are available is one major reason for the Reed-Solomon codes' popularity. One well-known method is due to Berlekamp and Massey. It is very commonly used in practice (see [McWS]). Other algorithms paralleling the Euclidean algorithm for the GCD of two polynomials have also been considered ([SKHN], [P]), and we will study that approach here.

For simplicity we will assume that the minimum distance of our Reed-Solomon code C is odd: $d = 2s + 1$. Then by Proposition 1, any s or fewer errors in a received word should be correctable.

Let $c = \sum_{j=0}^{q-2} c_j t^j$ be a codeword of C, in the polynomial representation from (2). In $\mathbb{F}_q[t]$, c is divisible by the generator polynomial

$$g = (t - \alpha)(t - \alpha^2) \cdots (t - \alpha^{d-1}).$$

Suppose that c is transmitted, but some errors are introduced, so that the received word is $r = c + e$ for some $e = \sum_{i \in L} e_i t^i$. L is called the set of *error locations*, and we assume $|L| \leq s$. The coefficients e_i are known as the *error values*.

Decoding Problem. *Given the received word r, determine the set of error locations L and the error values e_i for the error polynomial e with s or fewer nonzero terms (if such a polynomial exists).*

Once we find e, the decoding function will return $E^{-1}(r - e)$. To solve this problem we will proceed as follows. First, we can try to determine whether errors have occurred by computing the values of the polynomial form of the received word at $\alpha, \ldots, \alpha^{d-1}$. If $r(\alpha^j) = 0$ for all $j = 1, \ldots, d - 1$, then r is divisible by g, and assuming s or fewer errors have occurred, r must be the codeword we intended to

send. The values $s_j = r(\alpha^j)$ are called the *syndromes* of the received word. Note that

$$s_j = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j),$$

since c is a multiple of g. Hence the s_j are the values of the error polynomial for $j = 1, \ldots, d-1$.

The syndromes may be used as the coefficients in a polynomial

$$S(u) = \sum_{j=1}^{d-1} s_j u^{j-1},$$

called the *syndrome polynomial* for the received word r. Its degree is $d-2$ or less. By extending the definition of $s_j = e(\alpha^j)$ to *all* exponents j we can also consider the formal power series

$$\widehat{S}(u) = \sum_{j=1}^{\infty} s_j u^{j-1}.$$

Suppose we knew the error polynomial e for a received word with s or fewer errors. Then as noted above, $s_j = \sum_{i \in L} e_i (\alpha^j)^i = \sum_{i \in L} e_i (\alpha^i)^j$. By exchanging the order of summation, then summing formal geometric series, $\widehat{S}(u)$ can be written as

$$\begin{aligned} \widehat{S}(u) &= \sum_{j=1}^{\infty} s_j u^{j-1} \\ &= \sum_{i \in L} e_i \left(\sum_{j=1}^{\infty} (\alpha^i)^j u^{j-1} \right) \\ &= \sum_{i \in L} \frac{e_i \alpha^i}{(1 - \alpha^i u)} \\ &= \frac{w(u)}{\ell(u)}, \end{aligned} \tag{5}$$

where

$$\ell = \prod_{i \in L} (1 - \alpha^i u)$$

and

$$w = \sum_{i \in L} e_i \alpha^i \prod_{\substack{j \neq i \\ j \in L}} (1 - \alpha^j u).$$

The roots of ℓ are precisely the α^{-i} for $i \in L$. Since the error locations can be determined easily from these roots, ℓ is called the *error locator polynomial*. Turning to the numerator w, we see that

$$\deg w \leq \deg \ell - 1.$$

In addition, if $i \in L$,

$$w(\alpha^{-i}) = e_i \alpha^i \prod_{j \neq i, j \in L} (1 - \alpha^j \alpha^{-i}) \neq 0.$$

Hence w has no roots in common with ℓ. From this we deduce the important observation that the polynomials w and ℓ must be *relatively prime* (that is, their GCD is 1).

Similarly, if we consider the "tail" of the series $\widehat{S}$,

$$\begin{aligned}\widehat{S}(u) - S(u) &= \sum_{j=d}^{\infty} \left(\sum_{i \in L} e_i (\alpha^i)^j \right) u^{j-1} \\ &= u^{d-1} \cdot \frac{g(u)}{\ell(u)},\end{aligned} \tag{6}$$

where

$$g = \sum_{i \in L} e_i \alpha^{id} \prod_{\substack{j \neq i \\ j \in L}} (1 - \alpha^j u).$$

The degree of g is also at most $\deg \ell - 1$.

Combining (5) and (6), and writing $d - 1 = 2s$ we obtain the relation

$$w = \ell S + u^{2s} g. \tag{7}$$

The equation (7) is called the *key equation* for decoding.

The derivation of the key equation (7) assumed the error polynomial e was known. But now consider the situation in an actual decoding problem, assuming that no more than s errors occurred. Given the received word r, S is computed. Consider the key equation (7) as a relation between the known polynomials S, u^{2s}, and *unknowns* Ω, Λ, Γ:

$$\Omega = \Lambda S + u^{2s} \Gamma.$$

Suppose a solution $(\Omega, \Lambda, \Gamma)$ of the key equation is found, which satisfies the following *degree conditions*:

$$\begin{aligned}\deg \Lambda &\leq s \\ \deg \Omega &< \deg \Lambda,\end{aligned} \tag{8}$$

and in which Ω, Λ are relatively prime. We claim that in such a solution Λ must be a factor of $u^{q-1} - 1$, and its roots give the inverses of the error locations. This is a consequence of the following uniqueness statement.

(4.1) Theorem. *Suppose that s or fewer errors occur in the received word r, and let S be the corresponding syndrome polynomial. Up to a constant multiple, there exists a unique solution $(\Omega, \Lambda, \Gamma)$ of (7) that satisfies the degree conditions (8), and for which Ω and Λ are relatively prime.*

Proof. The existence of a solution follows from Algorithm (4.2) below. As above, the actual error locator ℓ and the corresponding w, g give one such solution. Let $(\Omega, \Lambda, \Gamma)$ be any other. Start with

$$\begin{aligned}w &= \ell S + u^{2s} g \\ \Omega &= \Lambda S + u^{2s} \Gamma,\end{aligned}$$

multiply the second by ℓ, the first by Λ and subtract. We obtain

$$w\Lambda = \Omega\ell + u^{2s}(g\Lambda - \ell\Gamma).$$

Since the degree conditions (8) are satisfied for both solutions, $w\Lambda$ and $\Omega\ell$ are actually polynomials of degree at most $2s-1$, so it follows that

$$w\Lambda = \Omega\ell$$

(and $g\Lambda = \ell\Gamma$). Since both pairs (w,ℓ) and (Ω,Λ) are relatively prime, they can differ only by a constant multiple. □

Given a solution of (7) for which the degree conditions (8) are satisfied, working backwards, we can determine the roots of $\Lambda = 0$ in $\mathbb{F}_q \setminus \{0\}$, and hence the error locations—if α^{-i} appears as a root, then $i \in L$ is an error location. Finally, the error values can be determined by the following observation.

Let (w,ℓ,g) be the solution of (7) in which the actual error locator polynomial ℓ (with constant term 1) appears. If $i \in L$, then

$$w(\alpha^{-i}) = \alpha^i e_i \chi_i(\alpha^{-i}) \tag{9}$$

where $\chi_i = \prod_{j\neq i}(1-\alpha^j u)$. (This is called the *Forney formula.*) Hence we can solve for e_i, once we know the error locations.

Theorem (4.1) and the preceding discussion show that solving the decoding problem can be accomplished by solving the key equation (7).

We will see that this can be done by adapting the Euclidean Algorithm for the GCD of two polynomials from §3.

(4.2) Algorithm. *The following algorithm will solve the key equation* (7) *and correctly decode* $RS(k,q)$*, provided the weight of the error is at most* $\lfloor\frac{d-1}{2}\rfloor = \lfloor\frac{q-k-1}{2}\rfloor$*. We assume* $d = 2s+1$*.*

Input : r, α primitive element

Output : e

FOR j FROM 1 TO $2s$ DO

$s_j := r(\alpha^j)$

$$S := \sum_{j=1}^{2s} s_j u^{j-1}$$

IF $S \neq 0$ THEN

$r_{-1} := u^{2s}; r_0 := S$

$g_{-1} := 1; g_0 := 0$

$\ell_{-1} := 0; \ell_0 := 1$

$k := 0$

WHILE $\deg(r_k) \geq s$ DO

divide r_k into $r_{k-1} : r_{k-1} = q_k r_k + r_{k+1}$

$g_{k+1} := g_{k-1} - q_k g_k$

$\ell_{k+1} := \ell_{k-1} - q_k \ell_k$

$k := k + 1$

IF $r_k \neq 0$ THEN

determine roots of $\ell_k(u) = 0$

find e_i using Forney Formula (9)

ELSE

$e := 0$

Note: The final values $(g_k(u), \ell_k(u))$ can be shown to be $(g(u), \ell(u))$ from (7), up to a constant multiple, and the final value r_k is w. So the roots of $\ell_k(u)$ are the inverses of the error locations.

We demonstrate this with an example. Use the field $\mathbb{F}_8$ (with $h(\alpha) = \alpha^3 + \alpha + 1$ as the problems below), and the Reed-Solomon code $RS(3, 8)$, which has $d = n - k + 1 = 7 - 3 + 1 = 5$, so $s = 2$. Suppose the codeword

$$c = ev(1) = (1, 1, 1, 1, 1, 1, 1)$$

is sent, but it is corrupted by errors to yield

$$r = (1, \alpha, 1, 1, 1, 1, \alpha^2 + 1),$$

or in polynomial form

$$r = 1 + \alpha t + t^2 + t^3 + t^4 + t^5 + (\alpha^2 + 1)t^6.$$

The first step is to compute the syndromes and the corresponding syndrome polynomial $S(u)$. For instance,

$$\begin{aligned} s_1 = r(\alpha) &= 1 + \alpha^2 + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6(\alpha^2 + 1) \\ &= 1 + (\alpha + 1) + (\alpha^2 + \alpha) + (\alpha^2 + \alpha + 1) + (\alpha^2 + \alpha + 1) \\ &= \alpha^2 \end{aligned}$$

Similarly,

$$s_2 = \alpha^4, s_3 = 0, s_4 = \alpha^4,$$

So

$$S(u) = \alpha^2 + \alpha^4 u + \alpha^4 u^3$$

(note the shift in indexing, as in the definition of S above).

We now begin the Euclidean algorithm to find the gcd of $u^{2t} = u^4$ and $S(u)$, keeping track of the remainders r_k, and g_k, ℓ_k. In the first division:

$$u^4 = \alpha^3 u S + (u^2 + \alpha^5 u)$$

so $q_0 = \alpha^3 u$ and $r_1 = u^2 + \alpha^5 u$. Hence

$$g_1 = g_{-1} - q_0 g_0 = 1 \qquad \ell_1 = \ell_{-1} - q_0 \ell_0 = \alpha^3 u.$$

We continue in the same way (only one more step is needed in the WHILE loop in (4.2) in this small example), and obtain the results collected in the following table

k	q_k	r_k	g_k	ℓ_k
-1		u^4	1	0
0	$\alpha^3 u$	$\alpha^4 u^3 + \alpha^4 u + \alpha^2$	0	1
1	$\alpha^4 u + \alpha^2$	$u^2 + \alpha^5 u$	1	$\alpha^3 u$
2		$\alpha^5 u + \alpha^2$	$\alpha^4 u + \alpha^2$	$u^2 + \alpha^5 u + 1$

Here $w = \alpha^5 u + \alpha^2$.

We stop here since $\deg(r_2) = 1 < 2 = s$. The next step is to find the roots of

$$\ell_2(u) = u^2 + \alpha^5 u + 1 = 0.$$

This can be done either by exhaustive search, or by factoring. We find

$$u^2 + \alpha^5 u + 1 = (1 + \alpha u)(1 + \alpha^6 u),$$

so the roots are $u = \alpha^6, \alpha$. But by the definition of the error locator polynomial, the locations of the errors are found from the inverses: $\alpha = \alpha^{-6}$ and $\alpha^6 = \alpha^{-1}$, so the errors occurred in locations 1 and 6. Finally we use the Forney Formula (9) to determine the error values e_1 and e_6: with $u = \alpha = \alpha^{-6}$, (9) says:

$$w(\alpha^{-1}) = \alpha e_1 \chi_1(\alpha^6) = \alpha e_1 (1 - \alpha^5) = \alpha^5 e_1.$$

Since $w(u) = \alpha^5 u + \alpha^2$, it follows that $w(\alpha^{-1}) = \alpha$, and so $e_1 = \alpha^3$. Similarly,

$$w(\alpha^{-6}) = \alpha^6 e_6 \chi_6(\alpha) = \alpha^6 e_6 (1 - \alpha^2) = \alpha^5 e_6$$

and $w(\alpha^{-6}) = 1$, and so $e_6 = \alpha^2$. Then

$$e(t) = \alpha^3 t + \alpha^2 t^6$$

and $r(u) + e(u) = \psi(c) = 1 + t + t^2 + \cdots + t^6$.

§5. Codes from Order Domains

A certain type of generalization of Reed-Solomon codes, known as *geometric Goppa codes* have been intensively studied in coding theory recently. These are named after their discoverer, V. D. Goppa. Some of these codes have extremely good parameters and the 1982 paper [TVZ] establishing this fact was a major landmark in the history of coding theory. The original formulation of the geometric Goppa codes required many notions from the classical theory of algebraic curves or function fields of transcendence degree one, as well as topics from number theory. However, there is a class of codes, including the most important geometric Goppa codes, for which a more elementary description is now available. We will introduce that treatment here and work out an example to give a brief first indication of how this works. A general reference for this is the article by Høholdt, van Lint and Pellikaan [HvLP] from the recently published *Handbook of Coding Theory*.

We will begin with some motivation. The construction of codes possessing good parameters and efficient decoding methods is the basic problem in coding theory. The Reed-Solomon codes are among the most powerful and successful codes for certain applications. Hence it is natural to try to generalize the construction of Reed-Solomon codes given above to produce other, potentially even better codes.

In the Reed-Solomon case, given an $f \in L_{k-1} = \text{Span}\{1, t, \dots, t^{k-1}\}$ for some $k < q$, we evaluated f at the nonzero elements of $\mathbb{F}_q$ to form the entries in a codeword of $RS(k, q)$. The set of nonzero elements of $\mathbb{F}_q$ is a collection of points on the affine line and L_{k-1} can be seen as a vector subspace of the ring $R = \mathbb{F}_q[t]$. A possible extension might proceed as follows. Let $S = \{P_1, \dots, P_n\}$ be a set of points in $\mathbb{F}_q^t$. We can then follow the Reed-Solomon case to define an evaluation mapping by

$$\begin{aligned} ev_S : \mathbb{F}_q[X_1, \dots, X_t] &\to \mathbb{F}_q^n \\ f &\mapsto (f(P_1), \dots, f(P_n)). \end{aligned}$$

The mapping ev_S is clearly linear, so if L is a finite-dimensional vector subspace of polynomials in t variables, the image $E = ev_S(L)$ will be a linear code in $\mathbb{F}_q^n$, called an *evaluation code*.

This gives a *very general* recipe for constructing codes, including the Reed-Solomon codes and a number of other classes of codes (such as the *Reed-Muller* codes, see [W]) that have been studied in coding theory. But there is no indication of how the S and the subspace L might be chosen to yield codes with good parameters and efficient decoding methods. It has recently become clear that one way to supply this missing ingredient is the notion of an order (or weight) function on a ring. The following is a slightly simplified treatment of a special case of this construction. Let $\Gamma \subset \mathbb{Z}_{\geq 0}$ be a subset of the form

$$\begin{aligned} \Gamma &= \langle m_1, \dots, m_t \rangle \\ &= \{a_1 m_1 + \cdots + a_t m_t : a_i \in \mathbb{Z}_{\geq 0}\}. \end{aligned}$$

Given a field F and a commutative ring R, we say that R is an *F-algebra* if R contains F and the multiplication in R restricted to F is the same as multiplication in F. For instance, the polynomial ring $\mathbb{F}_q[X_1, \dots, X_t]$ is an example of an $\mathbb{F}_q$-algebra.

(5.1) Definition. Let R be a finitely-generated commutative $\mathbb{F}_q$-algebra with identity, and let Γ be as above (for some choice of $m_1, \dots, m_t$). A surjective function

$\rho : R \to \{-\infty\} \cup \Gamma$ is said to be a *order function* on R if it satisfies the following properties for all $f, g \in R$, and $\lambda \in \mathbb{F}_q$:

(1) $\rho(f) = -\infty$ if and only if $f = 0$.
(2) $\rho(\lambda f) = \rho(f)$ for all $\lambda \neq 0$.
(3) $\rho(f+g) \preceq \max\{\rho(f), \rho(g)\}$, with equality if $\rho(f) \neq \rho(g)$.
(4) If $\rho(f) = \rho(g) \neq -\infty$, then there exists $\lambda \neq 0$ such that $\rho(f + \lambda g) \prec \rho(f)$.
(5) $\rho(fg) = \rho(f) + \rho(g)$.

The following Proposition develops some first properties of rings with order functions.

(5.2) Proposition. *Let ρ be a order function on R.*

(1) *We have $\rho(1) = 0$ (the additive identity in Γ), and hence $\rho(c) = 0$ for all $c \neq 0$ in $\mathbb{F}_q$.*
(2) *Every R having a order function is an integral domain. (A ring R is said to be an integral domain if it has no nonzero zero-divisors, where a zero-divisor is defined to be any element $a \in R$ such that $ab = 0$ for some nonzero $b \in R$.)*
(3) *Every set of elements of R with distinct ρ values is linearly independent over $\mathbb{F}_q$.*
(4) *There exists an $\mathbb{F}_q$-basis of R consisting of elements with distinct ρ values.*

As a first example, note that the ring $R = \mathbb{F}_q[X]$ satisfies this definition if we take $\rho(f) = \deg(f)$. Thus the Reed-Solomon code set-up is included in this larger context.

(5.3) Example. Let $q = 4$. In the collection of all polynomials in *two variables* X_1, X_2 with coefficients in $\mathbb{F}_4$, denoted $\mathbb{F}_4$, we consider the set R of all (finite) linear combinations of the monomials in the set

$$\Delta = \{X_1^i X_2^j : 0 \leq i \leq 2, j \geq 0\}.$$

As in our construction of the finite fields $\mathbb{F}_{p^r}$ before, we will make this set of monomials into a ring (not a field, though, since most elements will not have multiplicative inverses). We will do this by defining appropriate sum and product operations. The sum operation is the familiar sum of polynomials. For the product, we multiply the factors as polynomials, *then divide by the polynomial* $X_1^3 + X_2^2 + X_2$, thinking of the X_1^3 as the "leading term" for the division. For instance, dividing $X_1^3 + X_2^2 + X_2$ into $\alpha X_1^4 + X_1$ gives a quotient of αX_1 and a remainder of $\alpha X_2^2 + \alpha X_2 + X_1$. Notice that the terms that can appear in remainders are exactly the monomials in Δ. (In abstract algebra, the construction we are doing here leads to the idea of the *quotient ring*

$$\mathbb{F}_4[X_1, X_2]/I,$$

where $I = \langle X_1^3 + X_2^2 + X_2 \rangle$ is the *principal ideal* generated by $X_1^3 + X_2^2 + X_2$.)

For each monomial in Δ, we have the following notion of its *weight*, or order: $\rho(X_1^a X_2^b) = (2,3) \cdot (a,b) = 2a + 3b$. These are distinct because of the limitation $0 \leq a \leq 2$. So in fact the set Δ is an example of part (4) of Proposition (5.2) above. The set of integers we get here as ρ-values of monomials is precisely the subset

$$\Gamma = \langle 2, 3 \rangle \subset \mathbb{Z}$$

as in Definition (5.1) above. And in fact we can extend ρ to R by setting

$$\rho(f) = \max\{\rho(X_1^a X_2^b) : X_1^a X_2^b \text{ appears in } f\}.$$

We claim that this makes R into a order domain with $\Gamma = \langle 2, 3\rangle = \{2i + 3j : i, j \in \mathbb{Z}_{\geq 0}\} \subset \mathbb{Z}_{\geq 0}$.

We will now present the most important examples of evaluation codes as described at the start of this section. The ring $R = \mathbb{F}_q[t]$ leading to the Reed-Solomon codes can be thought of as the ring of polynomial functions on the line over the field $\mathbb{F}_q$. If R is an order domain, there is a geometric object associated to R in a similar way. Let $R = \mathbb{F}_q[X_1, \ldots, X_t]/I$ has an order function ρ. The geometric object associated to R is called the *variety* defined by the ideal I, the common zero locus of all polynomials in I:

$$X = \mathbf{V}(I) = \{(a_1, \ldots, a_t) \in \mathbb{F}_q^t : f(a_1, \ldots, a_t) = 0 \text{ for all } f \in I\}.$$

Since $\mathbb{F}_q$ is finite, X is also a finite set.

(5.4) Example. We continue with the order domain introduced in Example (5.3). To construct evaluation codes, we need to know the points in X, where $X = \mathbf{V}(X_1^3 + X_2^2 + X_2)$. There are exactly eight such points with coordinates in $\mathbb{F}_4$. Writing α for a primitive element of $\mathbb{F}_4$ (constructed using $h(\alpha) = \alpha^2 + \alpha + 1 = 0$), the eight points can be numbered as follows:

$$\begin{array}{lllllll} P_1 & = & (0,0) & & P_2 & = & (0,1) \\ P_3 & = & (1,\alpha) & & P_4 & = & (1,\alpha^2) \\ P_5 & = & (\alpha,\alpha) & & P_6 & = & (\alpha,\alpha^2) \\ P_7 & = & (\alpha^2,\alpha) & & P_8 & = & (\alpha^2,\alpha^2). \end{array}$$

To construct codes from an order domain, we will evaluate functions from some vector subspace L in R at the points in X. As in the Reed-Solomon case, the most useful vector subspaces L of R will have the form $L = \{f \in R : \rho(f) \leq m\}$ for some $m \in \Gamma$.

(5.5) Proposition. *Let (R, ρ) be an order domain as above. For $a \in \Gamma$, let $L_a = \{f \in R : \rho(f) \leq a\}$ and let $L_{-1} = \{0\}$. Then L_a is a finite dimensional vector subspace of R for all a.*

Proof. Closure of each L_a under sums and scalar multiples follows from parts (2) and (3) of Definition (5.1). □

We now present some examples of evaluation codes constructed using the L_a subspaces. We will write $E_a = ev_S(L_a)$, where ev_S is the evaluation mapping as above.

Continuing with R from (5.3) and (5.4) above, The code $E_3 = ev(V_3)$ is obtained as follows. The vector space L_3 is spanned by $\{1, X_1, X_2\}$, since $\rho(1) = 0$, $\rho(X_1) = 2$, $\rho(X_2) = 3$, and all other monomials in Δ have ρ-value at least 4. The codewords are obtained by evaluation at the eight points P_i above. This gives the following generator matrix for a code of block length $n = 8$ over $\mathbb{F}_4$:

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \end{pmatrix}. \tag{10}$$

Determining the minimum distance of the evaluation codes can be quite delicate, since it involves the subtle question of how many zeroes a polynomial in L_a can have *at the $\mathbb{F}_q$-rational points on X*. There are both geometric and arithmetic issues involved. In the following simple example the geometry suffices to understand what is going on.

Consider the E_3 code over $\mathbb{F}_4$ studied above. Each codeword is a linear combination of the three rows of the matrix G given in (10). Hence each codeword is formed by evaluation of some linear function $f = a+bX_1+cX_2$ at the 8 $\mathbb{F}_4$-rational points. We can use a famous result known as *Bézout's theorem* to give an upper bound for the number for the number of zero entries in a codeword, hence a lower bound for d. Bézout's theorem says that over any field, if X is a the set of zeroes of a polynomial F of degree m in two variables, and Y is the set of zeroes of a second polynomial G of degree n which has no factor of positive degree in common with F, then $|X \cap Y| \leq mn$.

Because X is defined by a polynomial of degree $m = 3$, it meets each line $\mathbf{V}(a+bX_1+cX_2)$ (degree $n = 1$) in at most $mn = 3 \cdot 1 = 3$ points, and hence $d \geq 5$. Some nonzero words in E_2 have weight exactly 5 since some of these lines intersect X in exactly 3 affine $\mathbb{F}_4$-rational points. The bound obtained from Bézout's theorem using the defining equations of X is sharp in this case, but that will not always be true.

Order domains and associated codes are a subject of current interest in coding theory. There is a very good decoding algorithm called the *Berlekamp-Massey-Sakata* (or BMS) algorithm that applies very generally. This construction looks quite promising as a way to construct codes with good parameters and efficient decoding methods. Although the Reed-Solomon codes have sufficed "so far" for the engineering problems we have encountered, many experienced coding theorists believe that it is only a matter of time before these more general codes are exploited also in the real world.

Problem Sessions

Problem 1. *(a) According to the Singleton bound, what is the largest possible number of codewords for a code $C \subset \mathbb{F}_2^8$ with minimum distance $d = 3$?*

(b) Improve this bound by showing that number of codewords in such a code is actually fewer than 32. (Hint: consider Hamming balls of radius 1 centered about the codewords.)

Problem 2. *Let $h(x)$ be an irreducible polynomial over $\mathbb{F}_p$ of degree r. Prove that $h(x)$ divides $x^{p^r-1} - 1$ Hint: Think about what part 5 in Theorem (2.1) says about the nonzero elements of the field $\mathbb{F}_{p^r}$.*

Problem 3. *Prove that if a polynomial of degree 3 or less has no roots, then it is irreducible. Is this true for polynomials of degree 4 and higher? Use the first statement to show that the polynomial $p(\alpha) = \alpha^3 + \alpha + 1$ is irreducible over $\mathbb{F}_2$.*

Problem 4. *Consider the field $\mathbb{F}_8$ obtained by using the irreducible polynomial $h(\alpha) = \alpha^3 + \alpha + 1$ from Problem 3. Write each of the elements of this field as a linear combination of $1, \alpha$, and α^2. Construct the addition and multiplication tables for the elements expressed in the form $0, 1, \alpha, \alpha^2, \ldots, \alpha^6$.*

Problem 5. *Show that* $h(\alpha) = \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1 = 0$ *is irreducible over* $\mathbb{F}_2$, *but* α *is not primitive for* $\mathbb{F}_{16}$.

Problem 6. *Find all of the irreducible, monic, quadratic polynomials over* $\mathbb{F}_3$. *Which of these polynomials have roots that are primitive for* $\mathbb{F}_9$*?*

Problem 7. *Prove that for* $a, b \in \mathbb{F}_{p^r}$, $(a+b)^{p^r} = a^{p^r} + b^{p^r}$. *(Hint: first show that* $\binom{p}{k}$ *is divisible by* p.*)*

Problem 8. *Consider the field* $\mathbb{F}_8$ *obtained by using the irreducible polynomial* $h(\alpha) = \alpha^3 + \alpha + 1$ *over* $\mathbb{F}_2$. *Let* C *denote the Reed-Solomon code over* $\mathbb{F}_8$ *with generator polynomial* $g(t) = (t-\alpha)(t-\alpha^2)(t-\alpha^3)(t-\alpha^4)$*?*

(a) What are the parameters of this code (i.e., what are n, k, *and* d*)?*

(b) Construct the generator matrix for this code.

(c) How would you encode the message consisting of the vector $(1, \alpha^3, \alpha^2)$, *i.e, what is the corresponding code vector* c*?*

(d) Suppose somebody hands you the vector $(\alpha^3, 1, \alpha^3, \alpha, \alpha, 1, 0)$ *and claims that it is a codeword. Verify that this is really the case without using the decoding algorithm. Can you determine the original message vector?*

Problem 9. *Consider the Reed-Solomon code from the previous problem. Suppose you receive the vector* $r = (1, \alpha^2, \alpha^4, \alpha^6, \alpha, \alpha, \alpha)$. *Assuming there are at most two errors, what is the original codeword?*

Problem 10. *Consider the Reed-Solomon code RS(k,q).*

(a) Prove that the rank of the generator matrix is k.

(b) Let $c \in \mathbb{F}_q^{q-1}$. *Prove the converse of Proposition (2.4), that is, if* $g(t)$ *divides* $\psi(c)$, *then* c *is a codeword.*

Problem 11. *What are the generator and parity check matrices for the Reed-Solomon code RS(3,8) where the polynomial* $h(\alpha) = \alpha^3 + \alpha^2 + 1$ *is used to generate* $\mathbb{F}_8$.

Problem 12. *Consider the Reed-Solomon code RS(k,q). Here is an alternative method of producing a map from the set of message vectors to the set of codewords. Let* $g(t)$ *be the generator polynomial for this code. For each message polynomial* $f(t)$, *define the corresponding codeword polynomial to be*

$$c(t) = t^{q-k-1} f(t) - d(t),$$

where $d(t)$ *is the remainder upon division of* $t^{q-k-1} f(t)$ *by* $g(t)$. *This method is called systematic encoding since the original message can be read directly from an uncorrupted codeword.*

(a) From Problem 2, we know $g(t)$ *divides* $t^{q-1} - 1$. *Define*

$$h(t) = \frac{t^{q-1} - 1}{g(t)},$$

which we call the parity-check polynomial. Show that $c(t)h(t)$ *is divisible by* $t^{q-1} - 1$ *if and only if* $c(t)$ *is a codeword.*

(b) Again, we consider the Reed-Solomon code RS(3,8). Given that

$$g(t) = (t-\alpha)(t-\alpha^2)(t-\alpha^3)(t-\alpha^4) = t^4 + \alpha^3 t^3 + t^2 + \alpha t + \alpha^3,$$

show how to encode the polynomials $f(t) = 1$ *and* $f(t) = t - \alpha$. *Note that this systematic coding does not produce the same results as the evaluation map ev* : $L_k \to \mathbb{F}_q^{q-1}$.

(c) Compute the parity-check polynomial for RS(3,8). Use this to test whether $c(t) = 1 + \alpha t + t^2 + \alpha t^3 + t^4 + \alpha t^5 + t^6$ *is a codeword.*

References

[HPL] T. Høholdt, R. Pellikaan, and J. van Lint, *Algebraic Geometry Codes*, Handbook of Coding Theory (W. Huffman and V. Pless, eds.), Elsevier, Amsterdam, 1998, pp. 871-962.

[HP] W. Cary Huffman, and V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge U. Press, Cambridge, UK.

[Imm] K. Immink, *Reed-Solomon Codes and the Compact Disc*, Reed-Solomon Codes and their Applications, S. Wicker and V. Bhargava, editors, IEEE Press, New York, NY, 1994, pp. 41–59.

[McES] R. McEliece and L. Swanson, *Reed-Solomon Codes and the Exploration of the Solar System*, Reed-Solomon Codes and their Applications, S. Wicker and V. Bhargava, editors, IEEE Press, New York, NY, 1994, pp. 25–40.

[McWS] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam.

[Mor] C. Moreno, *Algebraic Curves over Finite Fields*, Cambridge U. Press, Cambridge, UK.

[P] O. Pretzel, *Error-Correcting Codes and Finite Fields*, Oxford U. Press, Oxford, UK.

[SKHN] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, *A method for solving key equation for decoding Goppa codes*, Inform. and Control **27** (1975), 87–99.

[TVZ] M. Tsfasman, S.Vlăduţ, and T. Zink, *Modular Curves, Shimura Curves and Goppa Codes Better Than the Varshamov-Gilbert Bound*, Math. Nachr. **109** (1982), 21–28.

[vLi] J. van Lint, *Introduction to Coding Theory, 2nd edition*, Springer-Verlag, Berlin, 1992.

[W] S. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, NJ, 1995.

Department of Mathematics College of the Holy Cross Worcester, MA 01610
E-mail address: little@mathcs.holycross.edu

Department of Mathematics Loyola Marymount University Los Angeles, CA 90045
E-mail address: emosteig@lmu.edu