

Lecture Notes for Math 696

Coding Theory

Michael E. O'Sullivan
mosulliv@math.sdsu.edu
www-rohan.sdsu.edu/~mosulliv

March 4, 2002

1 Codes

Let F be a field and let F^n be a vector space of dimension n over F . We will write elements of F^n as row vectors. We define the *Hamming distance* function d on F^n by

$$d(x, y) = \#\{i : x_i \neq y_i\}$$

It is straightforward to prove that d is a metric (do it!); that is

$$\begin{aligned}d(x, y) &\geq 0 \quad \text{with equality iff } x = y \\d(x, y) &= d(y, x) \\d(x, y) + d(y, z) &\geq d(x, z)\end{aligned}$$

The *Hamming weight* of a vector is the number of nonzero positions in x ; so $\text{wt}(x) = d(x, 0)$. The *support* of x is the set of indices i for which x_i is nonzero.

An $[n, k, d]$ *linear code* C is a vector subspace of F^n of dimension k whose minimum distance is d . The *minimum distance* is defined by $d = \min\{\text{wt}(c) : c \in C \text{ and } c \neq 0\}$. We will usually say that C is a code and not mention the linearity, although it will always be assumed.

An $[n, k, d]$ code can be specified by a $k \times n$ *generator matrix* G , such that $C = \{uG : u \in F^k\}$. We will always assume that the rows of G are linearly independent so, the rows of G form a basis for C .

We can also specify C using a $n \times (n - k)$ *parity check matrix* H such that $C = \{c \in F^n : cH = 0\}$. Since the parity check matrix has $n - k$ columns, any $(n - k + 1)$ rows of H are linearly dependent. Thus for any choice of a $(n - k + 1)$ subset $I \subset \{1, \dots, n\}$ there is a codeword c whose support is contained in I . In particular, the minimum distance of C is at most $n - k + 1$. We have just derived the *Singleton bound*:

Proposition 1.1. *Let C be an $[n, k, d]$ linear code. $d \leq n - k + 1$.*

Definition 1.2. A code meeting the Singleton bound is called an *maximum distance separable* or MDS code.

The *dual code* of C is the code

$$C^\perp = \{v : v \cdot c = 0 \text{ for all } c \in C\}$$

If G is a generator matrix for C and H is a check matrix for C then H is a generator matrix for C^\perp and G is a check matrix for C^\perp .

Using a code for error correction

Let C be an $[n, k, d]$ code over \mathbb{F}_q with generator matrix G and check matrix H . I will sketch here the protocol for using C for error correction.

- We assume that information is presented to the *sender* as packets of k elements from \mathbb{F}_q . This is treated as a vector in \mathbb{F}_q^k .
- *Encoding* adds redundancy: $v = uG$ is in \mathbb{F}_q^n .
- Vector v is sent to the *receiver*.
- *Error* in the transmission channel is modeled as a vector $e \in \mathbb{F}_q^n$. The receiver gets $w = v + e$.
- *Decoding* is the attempt to recover v . Given w find the closest vector $c \in C$ to w . Using the triangle inequality, one can show that if $\text{wt } e < d/2$ then $c = v$.
- One v is found, matrix inversion of some $k \times k$ submatrix of G will produce u .

In practice, the final matrix inversion is avoided by using a *systematic* encoding process, that is one where u is a subvector of v . We can think of this as using a generator matrix which has a $k \times k$ submatrix which is the identity matrix (or some permutation of it).

As for decoding, the presumption is that the number of errors produced by the channel is very rarely more than $d/2$. The system designer usually has some idea, based on the electronics and experience, about the likelihood p_s of a symbol being in error. The code is chosen so that $d/2$ is “comfortably” larger than the expected number of symbol errors np_s .

2 Ideals

Definition 2.1. An *ideal* of a ring R is a subset $I \subset R$ which is closed under addition and closed under multiplication by an arbitrary element of R :

$$a + b \in I \quad \text{if } a, b \in I \tag{1}$$

$$ar \in I \quad \text{if } a \in I \text{ and } r \in R \tag{2}$$

The ideal I is *principal* if there is some $a \in I$ such that $I = \{ar : r \in R\}$. We say I is *generated by* a_1, a_2, \dots, a_s if $I = \{r_1a_1 + r_2a_2 + \dots + r_sa_s : r_i \in R\}$. We write $I = \langle a_1, a_2, \dots, a_s \rangle$.

Example 2.2. The principal ideals of \mathbb{Z} are multiples of a particular integer.

- (2) is the set of even numbers.
- (3) is the set of multiples of 3.
- (1) is all integers.
- (0) is just the set 0.

Proposition 2.3. *Every ideal in \mathbb{Z} is principal.*

PROOF: Let I be an ideal of \mathbb{Z} . If $I = \{0\}$, there is nothing to prove. Otherwise, let a be the smallest positive integer in I . Let b be any other nonzero element of I . Then by the properties of ideals, any linear combination of a and b is in I . Therefore $\gcd(a, b) \in I$. But the gcd of a and b is positive and less than or equal to a . Since a is the smallest positive element of I , we must have $\gcd(a, b) = a$. In other words an arbitrary element of I is divisible by a , so $I = \langle a \rangle$ is principal. \square

You can prove this one.

Proposition 2.4. *If $u \in I$ is a unit then $I = R$.*

Exercises 2.5.

- 1) Identify all the ideals of \mathbb{Z}/n .
- 2) Let F be a field. Show that every ideal in $F[x]$ is principal.
- 3) Identify all the ideals of $F[x]/f(x)$.
- 4) Find ideals in $\mathbb{R}[x, y]$ that are not principal.
- 5) The Chinese Remainder theorem says that if $n = \prod_{i=1}^s p_i^{a_i}$ with p_i distinct primes and $a_i > 0$ then

$$\mathbb{Z}/n \cong \mathbb{Z}/p_1^{a_1} \times \mathbb{Z}/p_2^{a_2} \times \cdots \times \mathbb{Z}/p_s^{a_s}$$

State and prove a similar result for the polynomial ring $F[x]$.

3 Reed-Solomon Codes

In this section I will give three different definitions of Reed-Solomon codes. We (well, you) will show that the definitions are equivalent. The reason to introduce all three perspectives is that each is useful for a purpose which the others do not satisfy well. One gives a direct description of the codewords, another is useful for decoding and the third is useful for encoding.

Throughout this section we will work over the field \mathbb{F}_q with α a primitive element of \mathbb{F}_q . The length of the code will always be $n = q - 1$ and k will designate the dimension of the code.

Reed-Solomon codes as evaluation codes

Let $RS_G(k)$ be the code with generator matrix

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \dots & \dots & \dots & \dots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \dots & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \dots & \dots & \dots & \alpha^{n-2} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{k-1} & \alpha^{2k-2} & \alpha^{3k-3} & \dots & \dots & \dots & \alpha^{n-k+1} \end{bmatrix} \quad (3)$$

More precisely, the matrix entries are $G_{ij} = \alpha^{ij}$ for $i = 0, \dots, k-1$ and $j = 0, \dots, n-1$

It is useful to think of this code as being obtained by evaluating polynomials at the field elements. Consider the non-zero field elements enumerated as α^j for j from 0 to $n-1$. The i th row of the G may be obtained by evaluating the polynomial x^i at the field elements α^j . The code itself can be described independently of the generator matrix as

$$RS_G(k) = \{(f(\alpha^0), f(\alpha^1), f(\alpha^2), \dots, f(\alpha^{n-1})) : f \in \mathbb{F}_q[x] \text{ and } \deg f < k\}$$

This gives a nice explicit description of the codewords.

Reed-Solomon codes as duals of evaluation codes

Let $RS_H(k)$ be the code with check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & \dots & \dots & 1 \\ \alpha & \alpha^2 & \alpha^3 & \dots & \dots & \alpha^{n-k} \\ \alpha^2 & \alpha^4 & \alpha^6 & \dots & \dots & \alpha^{2n-2k} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \alpha^{n-1} & \alpha^{n-2} & \alpha^{n-3} & \dots & \dots & \alpha^k \end{bmatrix} \quad (4)$$

Here we have constructed the check matrix by evaluating the polynomials x^i for $i = 1, \dots, k$ at the field elements $1, \alpha, \alpha^2, \dots, \alpha^{n-1}$. Notice the annoying fact that H is not quite the transpose of the generator matrix for $RS_G(n-k)$. We will remedy this in a later section §10.

It is now very easy to prove that the minimum distance of $RS_H(k)$ meets the Singleton bound, $n-k+1$. One simply checks that any $(n-k) \times (n-k)$ submatrix of H has nonzero determinant. This means that any $(n-k)$ rows of H are linearly independent. Therefore there can be no nonzero codeword whose support has $n-k$ or fewer positions.

In addition to establishing the minimum distance, this formulation of Reed-Solomon codes is useful for decoding.

Exercises 3.1. Prove that every $(n-k) \times (n-k)$ submatrix of H has nonzero determinant. (compare with a Vandermonde matrix.)

Reed-Solomon codes as cyclic codes

In this description of Reed-Solomon codes we use an association between vectors $(v_0, v_1, \dots, v_{n-1})$ and polynomials $v_0 + v_1x + v_2x^2 + \dots + v_{n-1}x^{n-1}$.

$$RS_C(k) = \{(c_0, c_1, \dots, c_{n-1}) : \sum_{j=0}^{n-1} c_j x^j \text{ vanishes at } \alpha, \alpha^2, \dots, \alpha^{n-k}\}$$

This formulation of Reed-Solomon codes is useful for encoding.

You can prove the following theorem as an exercise. We computed the minimum distance in the section on RS_H .

Theorem 3.2. $RS_G(k) = RS_H(k) = RS_C(k)$. The minimum distance of a $RS(k)$ meets the Singleton bound $n - k + 1$.

From now on we will just write $RS(k)$ for the Reed-Solomon code of dimension k

Exercises 3.3.

- 1) Prove the theorem.
- 2) Write Maple code to construct the matrices G and H for $RS(k)$ and check that the product is a matrix of zeros.

4 Cyclic Codes and Systematic Encoding

In this section we show the connection between cyclic codes of length n over F and ideals of $F[x]/(x^n - 1)$. We will show that there is a simple algorithm for encoding cyclic codes. We will also discuss the dual code to a cyclic code.

Definition 4.1. Let τ be the cyclic shift operator,

$$\tau(c_0, c_1, c_2, \dots, c_{n-1}) = (c_{n-1}, c_0, c_1, c_2, \dots, c_{n-2})$$

A code of length n over F is called *cyclic* if $\tau(c) \in C$ whenever $c \in C$.

Consider the ring $R = F[x]/(x^n - 1)$. We know from the section on polynomial rings that the set of polynomials of degree less than n is a system of representatives for R and that equivalence classes of $1, x, x^2, \dots, x^{n-1}$ form a basis for R as a vector space over F . The multiplicative structure of R is quite nice because $x(a_0 + a_1x + \dots + a_{n-1}x^{n-1}) = a_{n-1} + a_1x + a_2x^2 + \dots + a_{n-2}x^{n-1}$. We also showed in an exercise in §2 that any ideal of this ring is generated by some factor $g(x)$ of $x^n - 1$.

Proposition 4.2. *let $g(x)$ divide $x^n - 1$. Then the dimension of the ideal $\langle g(x) \rangle$ in $R = F[x]/(x^n - 1)$ as a vector space over F is $n - \deg g(x)$.*

PROOF: Let $g(x)h(x) = x^n - 1$ and let $d = \deg g(x)$. The polynomial $\sum_{i=0}^{n-d-1} a_i x^i g(x)$ has degree $\max\{i : a_i \neq 0\} \leq n - 1$. Its congruence class in R is therefore nonzero unless all the a_i are 0. This shows that $g(x), xg(x), \dots, x^{n-d-1}g(x)$ are linearly independent over F .

On the other hand,

$$x^{n-d}g(x) \equiv \sum_{i=1}^{n-d-1} h_i x^i g(x) \pmod{(x^n - 1)}$$

where $h(x) = x^{n-d} + \sum_{i=1}^{n-d-1} h_i x^i$. Thus $x^{n-d}g(x)$ is linearly dependent, modulo $x^n - 1$, on $x^i g(x)$ for $i = 0, 1, \dots, n-d-1$. One can show by similar methods that for $r \geq n-d$, $x^r g(x)$ is equivalent modulo $x^n - 1$ to a sum of $x^i g(x)$ $i = 0, 1, \dots, n-d-1$. Thus any element of $\langle g(x) \rangle$, say $a(x)g(x)$ can be written modulo $x^n - 1$ as a sum of $x^i g(x)$ for $i = 0, 1, \dots, n-d-1$. \square

Theorem 4.3. *Let C be a cyclic code of length n and dimension k over F . The set of polynomials $I = \{c_0 + c_1x + c_2x^2 + \dots + c_{n-1}x^{n-1} : (c_0, c_1, \dots, c_{n-1}) \in C\}$ is an ideal of $R = F[x]/(x^n - 1)$. Hence there is a unique codeword $(g_0, g_1, \dots, g_{n-k}, 0, 0, \dots, 0)$ in C with $g_{n-k} = 1$ such that a basis for C is $g, \tau g, \tau^2 g, \dots, \tau^{n-k-1} g$.*

Conversely, any ideal of $F[x]/(x^n - 1)$ gives rise to a cyclic code, and distinct ideals give distinct codes.

PROOF: Since C is closed under addition, the set I is also closed under addition. Since C is closed under multiplication by an element of F , so is I . Since C is closed under cyclic shift, τ , I is closed under multiplication by x . Now let $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ be an arbitrary element of R and let $c(x)$ in I , we seek to prove that their product is in I . From what we have shown about I , $x^i c(x) \in I$, so $a_i x^i c(x) \in I$, and finally, $(\sum_{i=0}^{n-1} a_i x^i) c(x) \in I$. This shows that I is closed under multiplication by an arbitrary element of R , and is therefore an ideal of R .

By an exercise in §2 any ideal of $F[x]/(x^n - 1)$ is generated by a factor of $x^n - 1$. Let $g(x)$ be the generator for I . The previous proposition says that the dimension of I is $n - \deg g(x)$. Since the map from C to I is one-to-one, and C has dimension k we have $k = n - d$ so $d = n - k$. Furthermore the proof of the proposition shows that $x^i g(x)$ for $i = 0 \dots n-d-1$ is a basis for I . Thus $g, \tau g, \tau^2 g, \dots, \tau^{n-k-1} g$ is a basis for C .

The proof that any ideal gives a cyclic code should be clear. \square

The polynomial g in the theorem is called the *generator polynomial* for C . One generator matrix for C is

$$\begin{bmatrix} g_0 & g_1 & g_2 & \dots & \dots & \dots & \dots & g_{n-k} & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & g_0 & g_1 & g_2 & \dots & \dots & \dots & \dots & g_{n-k} & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \dots & \dots & \dots & \dots & g_{n-k} & 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \dots & g_0 & g_1 & g_2 & \dots & \dots & \dots & \dots & g_{n-k} \end{bmatrix} \quad (5)$$

Encoding

Let C be a cyclic code and suppose that we want to encode the message vector $u = (u_0, \dots, u_{k-1})$. We can multiply u by G to get a codeword, $v = uG$ and send v . Let

$u(x) = \sum_{i=0}^{k-1} u_i x^i$. You should check that the components of v are, in appropriate order, the coefficients of the polynomial product $u(x)g(x)$. Using electronic circuitry it is much simpler to compute the polynomial product than to compute an arbitrary matrix product. See Blahut Ch. 6 for numerous approaches to hardware implementation.

The product $u(x)g(x)$ has the disadvantage of not being a systematic encoding of $u(x)$. Fortunately there does exist a method for systematic encoding that is just as simple. Let $q(x)$ and $r(x)$ be the quotient and remainder when $x^{n-k}u(x)$ is divided by $g(x)$:

$$x^{n-k}u(x) - r(x) = q(x)g(x) \tag{6}$$

The left-hand side of the equation is

$$u_{k-1}x^{n-1} + u_{k-2}x^{n-2} + \dots + u_1x^{k+1} + u_0x^k + r_{k-1}x^{k-1} + \dots + r_1x + r_0$$

Furthermore since the LHS is a multiple of $g(x)$ the associated vector is a codeword of C . So this polynomial division gives a systematic encoder. The circuitry for polynomial division is similar to polynomial multiplication and can be found in Blahut.

Refer back to the section on Reed-Solomon codes for the description of $RS_C(k)$ over the field \mathbb{F}_q with α a primitive element. You should see that $RS_C(k)$ is a cyclic code with generator polynomial $g(x) = \prod_{i=1}^{n-k} (x - \alpha^i)$. We can therefore systematically encode a Reed-Solomon code using polynomial division as described above.

The dual code of a cyclic code

Let C be a cyclic code of length n over a field F and let $g(x)$ be the generator polynomial for C . We know $g(x)$ is a factor of $x^n - 1$, so let us suppose that $g(x)h(x) = x^n - 1$. Let $h(x) = h_0 + h_1x + h_2x^2 + \dots + h_kx^k$, and consider the matrix

$$H = \begin{bmatrix} h_k & 0 & 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ h_{k-1} & h_k & 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ h_{k-2} & h_{k-1} & h_k & 0 & \dots & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ h_0 & h_1 & h_2 & h_3 & \dots & \dots & \dots & \dots & \dots \\ 0 & h_0 & h_1 & h_2 & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & h_0 & h_1 & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & h_0 & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \dots & \dots & h_0 & h_1 \\ 0 & 0 & 0 & 0 & \dots & \dots & \dots & 0 & h_0 \end{bmatrix} \tag{7}$$

Computing the product of G from (5) and H one finds that the product of the i th row of G and the j th column of H ($i = 0, \dots, k-1, j = 0, \dots, n-k-1$) is the coefficient of x^{k+j-i} in the product of $g(x)$ and $h(x)$ (see exercise below). Since $g(x)$ and $h(x)$ were chosen to have product $x^n - 1$, as we let i and j wander over the ranges given we always get 0. This is because $k + j - i$ is minimized when $i = k - 1$ and $j = 0$ for which we

get $k + j - i = 1$ and it is maximized when $i = 0$ and $j = n - k - 1$ for which we get $k + j - i = n - 1$. Clearly all the coefficients of $x^n - 1$ for powers of x between 1 and $n - 1$ are indeed 0.

Notice that the transpose of H is not in the same form as G , but is quite similar. In fact, if we define a polynomial $\bar{h}(x) = h_0x^k + h_1x^{k-1} + \cdots + h_{k-1}x + h_k$, then H^T coincides with the matrix for the generating polynomial $\bar{h}(x)$. This would seem to imply that $\bar{h}(x)$ is a factor of $x^n - 1$. Well, it is. Notice that $\bar{h}(x) = x^k h(1/x)$ and let $\bar{g}(x) = x^{n-k} g(1/x)$. Then

$$\begin{aligned} x^n - 1 &= -x^n((1/x)^n - 1) \\ &= -x^n(g(1/x)h(1/x)) \\ &= -(x^{n-k}g(1/x))(x^k h(1/x)) \\ &= -\bar{g}(x)\bar{h}(x) \end{aligned}$$

So $\bar{h}(x)$ is a factor of $x^n - 1$, and it is therefore the generator polynomial of a cyclic code. Since we defined generator polynomials to be monic, strictly speaking a constant multiple of $\bar{h}(x)$ is the generator polynomial.

Definition 4.4. Let $g(x)$ be a factor of $x^n - 1$ over the field F , and let $d = \deg g(x)$. The *reciprocal polynomial* of $g(x)$ (relative to $x^n - 1$) is $\bar{g}(x) = x^d g(1/x)$.

Proposition 4.5. Let $g(x)h(x) = x^n - 1$ over some field F . The dual of the cyclic code of length n with generator polynomial $g(x)$ is cyclic with generator polynomial $\bar{h}(x)$, the reciprocal polynomial of $h(x)$.

Exercises 4.6.

1) Explain why RS_C is a cyclic code. In the sections on decoding we will use the code $RS(n - k)^\perp$. What is the generator polynomial for $RS(n - k)^\perp$?

2) Identify all cyclic codes of lengths n over \mathbb{F}_2 for several values of n in the range $n = 5, \dots, 20$. For each n make a table showing the generator polynomial, the dimension of the code, and the polynomial generating the dual code. You can write the generator in factored form with variable exponents. How many distinct codes are there altogether?

3) Do the same over \mathbb{F}_4

4) Do the same over \mathbb{F}_3 .

5) Justify the claim in the subsection on dual codes that the product of the i th row of G and the j th column of H is the coefficient of x^{k+j-i} in the product of $g(x)$ and $h(x)$. One way to proceed is to think of h as being indexed from $-\infty$ to ∞ , with $h_i = 0$ for $i < 0$ or $i > k$. Similarly one extends g . Show the product of the i th row of G and the j column of H is

$$\sum_m g_m h_{k-m+j-i}$$

6) Let $\alpha \in \mathbb{F}_9$ satisfy $\alpha^2 = \alpha + 1$. We will use the code $RS(n - k)^\perp$ which has dimension k and whose generator polynomial you computed in exercise 1). Which such code has minimum distance 5? Systematically encode $x^2 + 1$ for this code.

7) Implement systematic encoding for Reed-Solomon codes in Maple.

5 Decoding of Reed-Solomon codes

Decoding can be broken down into 6 steps:

1. Compute the syndrome;
2. Compute the error-locator polynomial;
3. Compute the error-locations;
4. Compute the error values;
5. Compute the codeword;
6. Compute the information vector.

Before going into depth on individual steps, let us discuss some general issues. Let C be a Reed-Solomon code of dimension k and length $n = q - 1$ over \mathbb{F}_q . The generator matrix G , is assumed to be systematic. Let

u be the information vector;

$v = uG$ the codeword;

e the error vector;

$w = v + e$ the received vector.

Since G is systematic, some k components of v are the components of u . We call these the *information symbols*, the other $n - k$ components are called the *check symbols*. Clearly step 6) in the algorithm is trivial. We simply strip the check symbols from a codeword and we are left with the information symbols.

If $\text{wt } e < d/2$, the algorithm we will use produces e in step 4) and then v in step 5) and then u in step 6), so the information is successfully transmitted. We will prove this. If $\text{wt } e \geq d/2$, the decoder will never produce the correct codeword in step 5). It will either

- fail to produce output in one of the steps 2), 3), or 4); or
- produce output in each of the steps.

In the latter case, step 5) will produce a codeword c unequal to v , the user will unknowingly strip the check symbols from c and use the incorrect data. In the former case, the failure of the decoder to produce output will signal to the user that the received vector is too corrupted to be decoded. The user can ask for retransmission, not use the data, or use the information symbols from w in the understanding that some may be erroneous.

We will not concern ourselves any further with the case when $\text{wt } e \geq d/2$. We will focus on the individual steps of the algorithm and what they produce under the assumption that $\text{wt } e < d/2$. We already remarked that step 6) is trivial. Step 5) is simple, we subtract the output of step 4), the error vector e , from the received vector w to get $v = w - e$.

Step 1) is also simple. The *syndrome* of w is the vector $s = wH$. Computation of s is actually easier than a general matrix multiplication because of the nice structure of H in equation (4). If we interpret w as a polynomial, then wH is the row vector $(w(1), w(\alpha), w(\alpha^2), \dots, w(\alpha^{k-1}))$. For each column of H , the polynomial evaluation can be done efficiently with Horner's method (see the exercise below). In hardware this is implemented with an adder and a single constant multiplier (α^j for the j th column).

If w is a codeword, which occurs if $e = 0$, then the syndrome is the zero vector. In this case we proceed to step 6). If s is not zero, we know the error vector is nonzero and we proceed to step 2).

To summarize, steps 1), 5), and 6) of decoding are fairly simple. Steps 2), 3), and 4) are more complex, enough so that each will be treated in its own section.

Exercises 5.1.

1) Let $f(x)$ be a polynomial of degree m and let α be a constant. Horner's method for the computation of $f(\alpha)$ is iterative

$$\begin{aligned} c_0 &= f_m \\ c_i &= f_{m-i} + c_{i-1} * \alpha \end{aligned}$$

Prove inductively that $c_k = \sum_{i=m-k}^m f_i \alpha^{i-m+k}$. Conclude that $c_m = f(\alpha)$.

2) Implementation of Horner's method can be done with the following circuit. Verify that it works.

6 The Error Locator Polynomial

It will be convenient to make the following subtle change to our definition of Reed-Solomon codes. Recall that the generator matrix for $RS(k)$ and the check matrix for $RS(n - k)$ were almost transposes of each other. They differ only in the first and last rows. We will now switch the roles. Henceforth, the top row of G is $(1, \alpha, \alpha^2, \dots, \alpha^{n-1})$ and the left hand column of H is 1 in each entry.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & \dots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \dots & \alpha^{n-k-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \dots & \dots & \alpha^{2n-2k-2} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{n-1} & \alpha^{n-2} & \alpha^{n-3} & \dots & \dots & \alpha^{k+1} \end{bmatrix} \tag{8}$$

The generator polynomial for this version of $RS(k)$ is therefore $\prod_{i=0}^{n-k-1} (x - \alpha^i)$.

Definition 6.1. For a vector e we define the *locator polynomial* to be

$$f_e(x) = \prod_{i:e_i \neq 0} (x - \alpha^i) \quad (9)$$

Step 2) of the algorithm takes the syndrome vector $s = eH$ as input and produces the error-locator polynomial. Before we get into the details of the algorithm for step 2) it will be useful to broaden the notion of the syndrome.

Let $f(x) = f_0 + f_1x + f_2x^2 + \cdots + f_{n-k-1}x^{n-k-1}$ be any polynomial and let $f = (f_0, f_1, \dots, f_{n-k-1})$ be the associated vector. Then

$$Ha = \begin{bmatrix} 1 & 1 & 1 & \cdots & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \cdots & \alpha^{n-k-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \cdots & \alpha^{2(n-k-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \alpha^{n-1} & \alpha^{n-2} & \cdots & \cdots & \alpha^{k+1} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \cdots \\ \cdots \\ f_{n-k-1} \end{bmatrix} = \begin{bmatrix} f(1) \\ f(\alpha) \\ f(\alpha^2) \\ \cdots \\ \cdots \\ \cdots \\ f(\alpha^{n-1}) \end{bmatrix}$$

For an error vector e and syndrome vector s we have

$$\begin{aligned} sf &= eHf \\ &= e \begin{bmatrix} f(1) \\ f(\alpha) \\ f(\alpha^2) \\ \cdots \\ \cdots \\ \cdots \\ f(\alpha^{n-1}) \end{bmatrix} \\ &= \sum_{i=0}^{n-1} e_i f(\alpha^i) \end{aligned}$$

Notice that letting $f(x) = x^m$ we get $sf = \sum_{i=0}^{n-1} e_i (\alpha^i)^m = s_m$. This discussion motivates the following definition.

Definition 6.2. Given an error vector e we define the *syndrome map*

$$\begin{aligned} S_e : \mathbb{F}_q[x] &\rightarrow \mathbb{F} \\ f(x) &\mapsto \sum_{i=0}^{n-1} f(\alpha^i) \end{aligned}$$

Here is an important property of the error locator polynomial:

$$S_e(f_e) = \sum_{i=0}^{n-1} e_i f_e(\alpha^i) = 0$$

This is because $f_e(\alpha^i) = 0$ whenever $e_i \neq 0$. Furthermore, $S_e(x^r f_e(x)) = 0$ since $x^r f_e(x)$ also vanishes at all α^i such that $e_i \neq 0$.

Proposition 6.3. *Let e have weight t and suppose that $f(x)$ satisfies $S_e(x^r f(x)) = 0$ for $r = 0, 1, \dots, t-1$. Then $f(x)$ is a multiple of $f_e(x)$, the locator polynomial for e . In particular, if $\deg f(x) \leq t$ then $f(x)$ is a constant multiple of $f_e(x)$.*

PROOF: If $S_e(x^r f(x)) = 0$ for all $r = 0, \dots, t-1$ then $S_e(g(x)f(x)) = 0$ for any polynomial $g(x)$ with $\deg g(x) < t-1$. Suppose that $e_k \neq 0$. Let $g(x) = \prod_{\substack{e_i \neq 0 \\ i \neq k}} (x - \alpha^i)$.

Then $g(x)$ vanishes at all error positions except α^k . Consequently,

$$\begin{aligned} S_e(g(x)f(x)) &= \sum_{i=0}^{n-1} e_i g(\alpha^i) f(\alpha^i) \\ &= e_k g(\alpha^k) f(\alpha^k) \end{aligned}$$

Now $e_k \neq 0$, $g(\alpha^k) \neq 0$ and, since $\deg g(x) = t-1$ we know $S_e(f(x)g(x)) = 0$. This forces $f(\alpha^k) = 0$. Since k was chosen arbitrarily such that $e_k \neq 0$, we conclude that $f(\alpha_k) = 0$ for all k such that $e_k \neq 0$. Thus $f(x)$ is a multiple of $f_e(x)$. Finally, if $\deg f(x) = t$, then, since $\deg f_e(x) = t$, $f(x)$ must be a constant multiple of $f_e(x)$. \square

We now come to the algorithm for computing the error-locator polynomial.

The Berlekamp-Massey algorithm

Definition 6.4. Let $f(x) \in \mathbb{F}_q[x]$. Suppose r is such that $S_e(x^r f(x)) \neq 0$ but $S_e(x^i f(x)) = 0$ for $0 \leq i < r$. We define the span, fail and discrepancy of $f(x)$ to be

- $\text{span } f(x) = r$,
- $\text{fail } f(x) = \deg f(x) + r$,
- $\text{discr } f(x) = S_e(x^r f(x))$.

If there is no such r then $\text{span } f(x)$ and $\text{fail } f(x)$ are defined to be infinite.

We need two basic results to establish the validity of the Berlekamp-Massey algorithm. The first concerns the impact of the existence of a polynomial having span c on polynomials of degree c . The second is the basic idea in the update process in the Berlekamp-Massey algorithm.

Proposition 6.5. *Let $\deg f(x) = b$, $\text{span } f(x) = c$ and (as a consequence) $\text{fail } f(x) = b + c$. For any polynomial $g(x)$ of degree c , $\text{span } g(x) \leq b$ and $\text{fail } g(x) \leq c + b$.*

Furthermore, for any $g(x)$ of degree at most c , $\text{fail } g(x) \leq c + b$.

PROOF: We know that $S_e(x^i(f(x))) = 0$ for $i < c$ and that $S_e(x^c f(x)) \neq 0$. Let $g(x)$ be any polynomial of degree c . We want to show that $S_e(x^i g(x)) \neq 0$ for some $i \leq b$, or equivalently, that $S_e(h(x)g(x)) \neq 0$ for some polynomial of degree at most b .

Let $\alpha \in \mathbb{F}_q$ be the leading coefficient of $g(x)$. Then $\deg(g(x) - \alpha x^c) < c$. Therefore

$$\begin{aligned} S_e(f(x)g(x)) &= S_e(f(x)(g(x) - \alpha x^c)) + S_e(\alpha x^c f(x)) \\ &= \alpha S_e(x^c f(x)) \end{aligned}$$

This is nonzero, so $\text{span } g(x) \leq b$ and $\text{fail } g(x) \leq b + c$ as was to be proved.

For any i between 0 and c , $\deg x^i f(x) = b + i$, $\text{span } x^i f(x) = c - i$ and $\text{fail } x^i f(x) = b + c$. Therefore any polynomial of degree $c - i$ has fail at most $c + b$. This establishes the last statement of the theorem. \square

Proposition 6.6. *Suppose that $f(x)$ has span r and discrepancy μ and that $g(x)$ has span c and discrepancy ν . If $r \leq c$ then $h(x) = f(x) - (\mu/\nu)x^{c-r}g(x)$ has span larger than r .*

PROOF: Notice that $\text{span } x^{c-r}g(x) = r$ since

$$\begin{aligned} S_e(x^i x^{c-r}g(x)) &= S_e(x^{c-(r-i)}g(x)) \\ &= \begin{cases} 0 & \text{if } i < r \\ \nu & \text{if } i = r \end{cases} \end{aligned}$$

Since S_e is linear it is clear that $\text{span } h(x) \geq r$, and the coefficient of $g(x)$ is chosen so that we get cancellation of the discrepancies.

$$\begin{aligned} S_e(x^r(f(x) - (\mu/\nu)x^{c-r}g(x))) &= S_e(x^r f(x)) - (\mu/\nu)S_e(x^c g(x)) \\ &= \mu - (\mu/\nu)\nu \end{aligned}$$

This shows that $\text{span } h(x) > r$. \square

Here is the Berlekamp-Massey algorithm, for the code $RS(k)$ over \mathbb{F}_q with check matrix (8).

Data: Compute for each m , polynomials $f^{(m)}(x)$ and $g^{(m)}(x)$.

Initialization: For $m = -1$, $f^{(-1)}(x) = 1$ and $g^{(-1)}(x) = 0$.

Algorithm: For $m = 0$ to $n - k - 1$,

$$\begin{aligned} r &= m - \deg f^{(m-1)}(x) \\ \mu &= S_e(x^r f^{(m-1)}(x)) \\ c &= \deg f^{(m-1)}(x) - 1 \end{aligned}$$

If $r < \deg f^{(m)}(x)$ or $\mu = 0$

$$\begin{bmatrix} f^{(m)}(x) \\ g^{(m)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\mu x^{c-r} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} f^{(m-1)}(x) \\ g^{(m-1)}(x) \end{bmatrix}$$

Otherwise,

$$\begin{bmatrix} f^{(m)}(x) \\ g^{(m)}(x) \end{bmatrix} = \begin{bmatrix} x^{r-c} & -\mu \\ \mu^{-1} & 0 \end{bmatrix} \begin{bmatrix} f^{(m-1)}(x) \\ g^{(m-1)}(x) \end{bmatrix}$$

Output $f^{(n-k-1)}(x)$.

Theorem 6.7. *At the end of the m th iteration,*

1. $f^{(m)}(x)$ is monic and $\text{fail } f^{(m)}(x) > m$
2. $g^{(m)}(x)$ is either 0 or

$$\begin{aligned} \text{span } g^{(m)}(x) &= \deg f^{(m)}(x) - 1, \\ \text{fail } g^{(m)}(x) &\leq m, \text{ and } \text{discr } g^{(m)}(x) = 1 \end{aligned}$$

PROOF: We proceed by induction. The initial case, $m = -1$ is easily verified. Assume the statements of the theorem are true for $m - 1$; we will prove them for m .

If $\mu = 0$ then $f^{(m-1)}(x)$ satisfies $\text{fail } f^{(m-1)}(x) > m$, so the algorithm sets $f^{(m)}(x) = f^{(m-1)}(x)$ and $g^{(m)}(x) = g^{(m-1)}(x)$. Item 2) of the proposition is true by the induction hypothesis.

If $\mu \neq 0$ but $r < \deg f^{(m-1)}(x)$, then the algorithm sets

$$f^{(m)}(x) = f^{(m-1)}(x) - \mu x^{c-r} g^{(m-1)}(x)$$

where $c = \deg f^{(m-1)}(x) - 1$. By Proposition 6.6, $\text{span } f^{(m)}(x) > r$. We will show next that $\deg f^{(m)}(x) = \deg f^{(m-1)}(x)$, so $\text{fail } f^{(m)}(x) > m$ as desired.

By the induction hypothesis on $g^{(m)}(x)$,

$$\begin{aligned} \deg g^{(m-1)}(x) &= \text{fail } g^{(m-1)}(x) - \text{span } g^{(m-1)}(x) \\ &\leq m - 1 - (\deg f^{(m-1)}(x) - 1) \\ &\leq m - \deg f^{(m-1)}(x) \end{aligned}$$

Therefore we have

$$\begin{aligned} \deg x^{c-r} g^{(m-1)}(x) &\leq (f^{(m-1)}(x) - 1) - (m - f^{(m-1)}(x)) + m - \deg f^{(m-1)}(x) \\ &= \deg f^{(m-1)}(x) - 1 \end{aligned}$$

This shows that $\deg f^{(m)}(x) = \deg f^{(m-1)}(x)$ and also that $f^{(m)}(x)$ is monic.

Finally, suppose that $\mu \neq 0$ and $r \geq \deg f^{(m-1)}(x)$. Similar computations to those in the preceding case show that $f^{(m)}(x) = x^{r-c} f^{(m-1)}(x) - \mu g^{(m-1)}(x)$ is monic, of degree $r + 1$ and with $\text{fail } f^{(m)}(x) > m$. Furthermore, $g^{(m)}(x) = \mu^{-1} f^{(m-1)}(x)$ has span r , fail m , and discrepancy 1. \square

Remark 6.8. Since $\text{span } g^{(m)}(x) = \deg f^{(m)}(x) - 1$ and $\text{fail } g^{(m)}(x) \leq m$, Proposition 6.5 tells us that no polynomial of degree less than $\deg f^{(m)}(x)$ has fail larger than m . We know that $\text{fail } f_e(x) > m$ for all m , so $\deg f^{(m)}(x) \leq \deg f_e(x) = \text{wt } e$.

In the algorithm, the case where $r \geq \deg f^{(m)}$ and $\mu \neq 0$ leads to a change of degree in the f polynomial. The reason for the change is evident from the remark, if $\text{span } f^{(m)}(x) \geq \deg f^{(m)}(x)$, then no polynomial of degree $\deg f^{(m)}(x)$ can have fail larger than m .

Corollary 6.9. For $m \geq 2t - 1$, $f^{(m)}(x) = f_e(x)$. Therefore, using the Berlekamp-Massey algorithm for decoding the code $RS(k)$ whose minimum distance is $d = n - k + 1$, the error locator can be found provided $\text{wt } e \leq (d - 1)/2$.

PROOF: Let $m \geq 2t - 1$. By the remark, $\deg f^{(m)}(x) \leq t$. Since $\deg f^{(m)}(x) > m$, $S_e(f(x)) = 0$ for $r \leq m - \deg f^{(m)}(x)$. But $m - \deg f^{(m)}(x) \geq 2t - 1 - t = t - 1$. Now, Proposition 6.3 tells us that $f_e(x)$ divides $f^{(m)}(x)$. Since $f^{(m)}(x)$ is monic of degree at most t we have $f^{(m)}(x) = f_e(x)$.

For $RS(k)$ with check matrix H in (8), the algorithm iterates from 0 to $n - k - 1$. If the number of errors is t and $t \leq (d - 1)/2$ then

$$\begin{aligned} 2t - 1 &\leq d - 2 \\ &= n - k - 1 \end{aligned}$$

so $f^{(n-k-1)}$ is the error locator. □

Exercises 6.10.

- 1) Fill in the details in the proof of Theorem 6.7 for $\mu \neq 0$ and $r \geq \deg f^{(m)}(x)$.
- 2) Let $\alpha \in \mathbb{F}_9$ satisfy $\alpha^2 = \alpha + 1$. Consider the RS code over \mathbb{F}_9 with $k = 4$ and check matrix from (8). Compute the syndrome and apply the Berlekamp-Massey algorithm for the received vector $w = (\alpha^7, 0, 1, \alpha, \alpha^4, \alpha^4, \alpha^3, \alpha^6)$.
- 3) Let $\alpha \in \mathbb{F}_{16}$ satisfy $\alpha^4 = \alpha + 1$. Consider the RS code over \mathbb{F}_{16} with $k = 9$ and check matrix from (8). Compute the syndrome and apply the Berlekamp-Massey algorithm for the received vector $w = (1, 1, \alpha^2, \alpha^{11}, 1, \alpha^7, 1, \alpha^6, \alpha^8, \alpha, \alpha^9, \alpha^5, 1, \alpha^6, \alpha^7)$

7 Finding the Error Locations

Step 3) of the algorithm is fairly simple. The input is the error-locator polynomial $f_e(x)$. The output is a list of the error locations. Since the error locations are the roots of the polynomial $f_e(x)$, we need only evaluate $f_e(x)$ at all the field elements α^i . We define a vector \tilde{e} by

$$\tilde{e}_i = \begin{cases} 1 & \text{if } f_e(\alpha^i) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Efficient evaluation of $f_e(\alpha^i)$ can be done using Horner's method and implemented in circuitry with an adder and constant multiplier.

Exercises 7.1.

- 1) Find the error locations for problem 2) of Section 6.
- 2) Find the error locations for problem 3) of Section 6.

8 The Error Evaluator Polynomial

Step 4) of the algorithm takes as input the error-locator polynomial $f_e(x)$, the list of locations \tilde{e} , and the syndrome vector s . Recall that we defined the syndrome vector for a error vector e and later introduced the syndrome map $S_e : \mathbb{F}_q[x] \rightarrow \mathbb{F}_q$. We expand the

notion of the syndrome a bit further by defining a syndrome polynomial. Recall that $n = q - 1$, and that for each nonzero $\beta \in \mathbb{F}_q$, $(x - \beta)$ is a factor of $x^n - 1$. Note that

$$\frac{x^n - 1}{x - \beta} = x^{n-1} + \beta x^{n-2} + \beta^2 x^{n-3} + \cdots + \beta^{n-2} x + \beta^{n-1}$$

For a vector e , consider the weighted sum of polynomials like those above:

$$\begin{aligned} \sum_{i=0}^{n-1} e_i \frac{x^n - 1}{x - \alpha^i} &= \sum_{i=0}^{n-1} e_i \sum_{m=0}^{n-1} x^{n-1-m} (\alpha^i)^m \\ &= \sum_{m=0}^{n-1} x^{n-1-m} \sum_{i=0}^{n-1} e_i (\alpha^i)^m \\ &= \sum_{m=0}^{n-1} x^{n-1-m} s_m \end{aligned}$$

where $s_m = \sum_{i=0}^{n-1} e_i (\alpha^i)^m$ is a coordinate of the syndrome vector.

Definition 8.1. For $m = 0, 1, \dots, n - 1$ let $s_m = S_e(x^m)$. The *syndrome polynomial* associated to e is

$$s(x) = s_0 x^{n-1} + s_1 x^{n-2} + \cdots + s_{n-2} x + s_{n-1}$$

Here is another characterization of the error locator polynomial, obtained from the syndrome polynomial.

Proposition 8.2. Let $s(x)$ be the syndrome polynomial associated to e and let $f(x)$ be an arbitrary polynomial. Then $f(x)s(x)$ is in the ideal generated by $x^n - 1$ if and only if $f(x)$ is a multiple of $f_e(x)$.

PROOF: We have

$$s(x) = (x^n - 1) \sum_{m=0}^{n-1} \frac{e_i}{x - \alpha^i}$$

Therefore

$$\begin{aligned} f_e(x)s(x) &= \left(\prod_{j: e_j \neq 0} (x - \alpha^j) \right) (x^n - 1) \sum_{m=0}^{n-1} \frac{e_i}{x - \alpha^i} \\ &= (x^n - 1) \left[\sum_{m=0}^{n-1} e_i \prod_{\substack{j: e_j \neq 0 \\ j \neq i}} (x - \alpha^j) \right] \end{aligned}$$

This computation shows that $f_e(x)s(x)$ is in the ideal generated by $x^n - 1$.

Conversely, suppose that $f(x)s(x) \in \langle x^n - 1 \rangle$. Then for each i such that $e_i \neq 0$, $f(x)s(x) \equiv 0 \pmod{x - \alpha^i}$. But

$$s(x) \equiv \prod_{\substack{j: e_j \neq 0 \\ j \neq i}} (x - \alpha^j) \pmod{x - \alpha^i}$$

which is nonzero. Thus we must have $f(x)$ congruent to 0 mod $x - \alpha^i$. Since i was arbitrary, $f(x)$ is divisible by $x - \alpha^i$ for each error location i . Therefore $f(x)$ is divisible by $f_e(x)$. \square

Definition 8.3. Let $s(x)$ be the syndrome polynomial for e . Let $\phi_e(x)$ be the polynomial such that $f_e(x)s(x) = \phi_e(x)(x^n - 1)$. We call $\phi_e(x)$ the *evaluator polynomial* for e .

The reason we call $\phi_e(x)$ the error evaluator is that it can be used to evaluate errors. Before seeing how that works we need to define the derivative of a polynomial.

The derivative of a polynomial

Over a finite field we cannot define the derivative using limits because we have no way to let the distance between two field elements go to zero. Nevertheless, over any field we can define the derivative using the usual formula.

Definition 8.4. Let $f(x) = f_0 + f_1x + f_2x^2 + \dots + f_dx^d$ be a polynomial defined over a field F . The derivative of $f(x)$ is

$$(f(x))' = f_1 + 2f_2x + 3f_3x^2 + \dots + df_dx^{d-1}$$

The integer coefficient k of f_kx^{k-1} is the k -fold sum of 1_F , so it is computed modulo the characteristic of F . We will also use the simpler notation $f'(x)$.

We must verify that the usual rules for differentiation hold.

Theorem 8.5. *The derivative satisfies the following properties. For $f(x), g(x) \in F[x]$ and $\alpha \in F$,*

- $(f(x) + g(x))' = f'(x) + g'(x)$ (*Linearity*)
- $(\alpha f(x))' = \alpha f'(x)$ (*Homogeneity*)
- $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$ (*Leibnitz' rule*)

PROOF: The proofs for linearity and homogeneity are easy. The monomial case of Leibnitz' rule follows:

$$\begin{aligned} (x^i x^j)' &= (x^{i+j})' \\ &= (i+j)x^{i+j-1} \\ &= ix^{i-1}x^j + x^i(jx^{j-1}) \\ &= (x^i)'x^j + x^i(x^j)' \end{aligned}$$

We use the monomial case to derive Leibnitz' rule for a monomial times an arbitrary polynomial. One can easily extend to the product of two arbitrary polynomials.

$$\begin{aligned}
(x^i g(x))' &= \sum_j g_j (x^i x^j)' \\
&= \sum_j g_j ((x^i)' x^j + x^i (x^j)') \\
&= (x^i)' \sum_j g_j x^j + x^i \sum_j g_j (x^j)' \\
&= (x^i)' g(x) + x^i g'(x)
\end{aligned}$$

□

Computing the error values

We now return to step 4) of decoding, computing the error values. The following proposition shows that we need only evaluate the two polynomials $f'_e(x)$ and $\phi_e(x)$ at the error locations. The proof is left as an exercise.

Proposition 8.6. *Let $f_e(x)$ and $\phi_e(x)$ be the locator and evaluator polynomial for the error vector e . For each error position k ,*

$$e_k = \frac{\phi_e(\alpha^k)}{f'_e(\alpha^k)}$$

Computing $f'_e(x)$ is a simple matter from the definition of the derivative. Computing $\phi_e(x)$ is basically a polynomial multiplication. Since we know that $f_e(x)s(x) = \phi_e(x)(x^n - 1)$, and $\deg f_e(x) = t$, we have $\deg \phi_e(x) = \deg f_e(x) + \deg s(x) - n \leq t - 1$. Furthermore, the coefficient of x^j in $\phi_e(x)$ is the coefficient of x^{n+j} in $f_e(x)s(x)$. This coefficient is

$$f_{j+1}s_0 + f_{j+2}s_1 + f_{j+3}s_2 + \cdots + f_{t-1}s_{t-2-j} + f_t s_{t-1-j}$$

where $f_e(x) = f_0 + f_1 x + \cdots + f_t x^t$.

It is worth pointing out that the decoder can only compute the first few terms of $s(x)$ —namely $s_m x^m$ for $m \leq n - k - 1$. At first glance it seems that we cannot hope to compute $\phi_e(x)$, since it is the product of $f_e(x)$ and $s(x)$. But, the previous paragraph shows that we only need s_m for $m \leq t - 1$ to compute $\phi_e(x)$. Since $t < d/2$ and $d = n - k + 1$, the decoder does have enough of the s_m to do this.

Exercises 8.7.

- 1) Prove that for $\alpha_i \in F$ and $f(x) = \prod_{i=1}^r (x - \alpha_i)$ that $f'(x) = \sum_{i=1}^r \prod_{j \neq i} (x - \alpha_j)$.
- 2) With $f(x)$ as above, show that $f'(\alpha_i) = \prod_{j \neq i} (\alpha_i - \alpha_j)$.
- 3) Prove Proposition 8.6.
- 4) Find the error values for problem 2) of Section 6
- 5) Find the error values for problem 3) of Section 6

9 Other Approaches to Error Evaluation

One drawback of the method for error evaluation described in the previous section is that the computation of the evaluator polynomial introduces a delay between the computation of the locator polynomial and the computation of the error values. We show in this section that the Berlekamp-Massey algorithm can be modified to compute the error evaluator polynomial as well as the error locator polynomial. Using this extension of the Berlekamp-Massey algorithm eliminates the aforementioned delay, but it introduces a different practical problem. It requires extra storage to handle two new polynomials $\phi^{(m)}(x)$ and $\psi^{(m)}(x)$. Fortunately, an even better method will be derived from the expanded BM algorithm that eliminates the need to compute the evaluator polynomial.

Here is the Berlekamp-Massey algorithm, for the code $RS(k)$ over \mathbb{F}_q with check matrix (8).

Data: Compute for each m , a matrix of polynomials

$$B^{(m)} = \begin{bmatrix} f^{(m)}(x) & \phi^{(m)}(x) \\ g^{(m)}(x) & \psi^{(m)}(x) \end{bmatrix}$$

Initialization: For $m = -1$,

$$B^{(-1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Algorithm: For $m = 0$ to $n - k - 1$,

$$r = m - \deg f^{(m-1)}(x)$$

$$\mu = S_e(x^r f^{(m-1)}(x))$$

$$c = \deg f^{(m-1)}(x) - 1$$

If $r < \deg f^{(m)}(x)$ or $\mu = 0$, set

$$U^{(m)} = \begin{bmatrix} 1 & -\mu x^{c-r} \\ 0 & 1 \end{bmatrix}$$

Otherwise, set

$$U^{(m)} = \begin{bmatrix} x^{r-c} & -\mu \\ \mu^{-1} & 0 \end{bmatrix}$$

Then,

$$B^{(m)} = UB^{(m-1)}$$

Output $f^{(n-k-1)}(x)$ and $\phi^{(n-k-1)}(x)$.

The polynomials $f^{(m)}(x)$ and $g^{(m)}(x)$ in the new algorithm are clearly the same as those produced by the original one. Before proving the algorithm produces $\phi_e(x)$ we need the following proposition.

Proposition 9.1. *For $0 \leq r \leq n-1$, $S_e(x^r f(x))$ is the coefficient of x^{n-1-r} in $f(x)s(x)$.*

PROOF: Let $f(x) = f^0 + f_1x + f_2x^2 + \dots + f_dx^d$. Then

$$S_e(f(x)) = f_0s_0 + f_1s_1 + \dots + f_ds_d$$

This is also the coefficient of x^{n-1} in $f(x)s(x)$. Likewise $S_e(x^r f(x))$ is the coefficient of x^{n-1} in $x^r f(x)s(x)$ and therefore is also the coefficient of x^{n-1-r} in $f(x)s(x)$. \square

Theorem 9.2. *At the end of the m th iteration of the algorithm,*

1. $f^{(m)}(x)s(x) - \phi^{(m)}(x)(x^n - 1)$ has degree at most $n - m + \deg f^{(m)}(x) - 1$
2. $g^{(m)}(x)s(x) - \psi^{(m)}(x)(x^n - 1)$ is monic and has degree exactly $n - \deg f^{(m)}(x)$.

PROOF: We proceed by induction. When $m = -1$, item 1) says that $\deg s(x) \leq n - 1$, and item 2) says that $x^n - 1$ is monic of degree n , and both statements are true. Assume the statements of the theorem are true for $m - 1$; we will prove them for m .

In the algorithm we set $\mu = S_e(x^r f^{(m-1)}(x))$ where $r = m - f^{(m-1)}(x)$. By the proposition, μ is the coefficient of $x^{n-1-m+\deg f^{(m-1)}(x)}$ in $f(x)s(x)$. If $\mu = 0$, both items of the proposition are satisfied by setting $B^{(m)} = B^{(m-1)}$.

If $\mu \neq 0$ and $r < \deg f^{(m)}(x)$ then

$$\begin{aligned} f^{(m)}(x) &= f^{(m-1)}(x) - \mu x^{c-r} g^{(m-1)}(x) \\ \phi^{(m)}(x) &= \phi^{(m-1)}(x) - \mu x^{c-r} \psi^{(m-1)}(x) \end{aligned}$$

Therefore we get,

$$\begin{aligned} f^{(m)}(x)s(x) - \phi^{(m)}(x)(x^n - 1) &= \left[f^{(m-1)}(x)s(x) - \phi^{(m-1)}(x)(x^n - 1) \right] \\ &\quad - \mu \left[x^{c-r} \left(g^{(m-1)}(x)s(x) - \psi^{(m-1)}(x)(x^n - 1) \right) \right] \end{aligned} \quad (10)$$

By the induction hypothesis, both bracketed terms have degree $n - m + \deg f^{(m-1)}(x)$. The factor μ ensures the leading terms cancel. Thus $f^{(m)}(x)s(x) - \phi^{(m)}(x)(x^n - 1)$ has degree at most $n - 1 - m + \deg f^{(m)}(x)$ as required.

The case when $r \geq \deg f^{(m)}(x)$ is similar. \square

Corollary 9.3. *When $m \geq 2t - 1$, $\phi^{(m)}(x) = \phi_e(x)$.*

PROOF: For $m \geq 2t - 1$, Corollary 6.9 says that $f^{(m)}(x) = f_e(x)$. We also know that $f^{(m)}(x)s(x) = \phi_e(x)(x^n - 1)$. Therefore,

$$\begin{aligned} f^{(m)}(x)s(x) - \phi^{(m)}(x)(x^n - 1) &= \phi_e(x)(x^n - 1) - \phi^{(m)}(x)(x^n - 1) \\ &= (\phi_e(x) - \phi^{(m)}(x))(x^n - 1) \end{aligned}$$

The theorem says that $f^{(m)}(x)s(x) - \phi^{(m)}(x)(x^n - 1)$ has degree at most $n - m + t - 1 < n$. This can only occur if $\phi_e(x) = \phi^{(m)}(x)$. \square

Error evaluation without the evaluator polynomial

The real benefit of the previous algorithm is not that it is the most practical, but that it gives a new formula for computing error values that does not require the error evaluator $\phi_e(x)$.

From the algorithm it is clear that

$$B^{(m)} = \left(\prod_{i=0}^m U^{(i)} \right) B^{(-1)}$$

Since $B^{(-1)}$ is the identity matrix and each $U^{(i)}$ has determinant 1 we get

$$\begin{aligned} \det B^{(m)} &= f^{(m)}(x)\psi^{(m)}(x) - g^{(m)}(x)\phi^{(m)}(x) \\ &= 1 \end{aligned}$$

In particular, when $m \geq 2t+1$ we see that $g^{(m)}(x)$ and $\psi^{(m)}(x)$ are two polynomials that give a combination of $f_e(x)$ and $\phi_e(x)$ which is 1. Let α^i be an error position. Then evaluating at α^i we get $g^{(m)}(\alpha^i)\phi_e(\alpha^i) = -1$. From Proposition 8.6 we have $\phi_e(\alpha^i) = e_i f'_e(\alpha^i)$. Substituting in the previous equation we have established the following proposition.

Proposition 9.4. *Let $m > 2t - 1$ and let $g^{(m)}(x)$ be as in the Berlekamp-Massey algorithm. If $e_i \neq 0$ then*

$$e_k = -(f'_e(\alpha^i)g^{(m)}(\alpha^i))^{-1} \tag{11}$$

Exercises 9.5.

- 1) Fill in the details in the proof of Theorem 9.2 for $\mu \neq 0$ and $r \geq \deg f^{(m)}(x)$.
- 2) Find the error values for problem 2) of Section 6.
- 3) Find the error values for problem 3) of Section 6.

10 Other Reed-Solomon Codes and Related Codes

We have defined Reed-Solomon codes in the strictest manner, using a particular pair of generator/check matrices. Actually we altered the original definition of the codes when we looked at decoding. The basic ideas behind the construction that we used can be modified in several ways to give similar codes, which are also called Reed-Solomon codes. In this section we look at this broader class of codes. We also consider subfield subcodes of Reed-Solomon codes.

Extended Reed-Solomon codes

Any code C of length n can be extended to a code \tilde{C} of the same dimension and of length $n + 1$ as follows. The code vectors of \tilde{C} are (c_0, c_1, \dots, c_n) where $(c_1, \dots, c_n) \in C$ and $c_0 = -(c_1 + c_2, \dots, c_n)$. If H is a check matrix for C then a check matrix for \tilde{C} is

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

If the original code had minimum distance d the new code will usually have distance $d + 1$. It may have distance d if the sum of the components of any codeword of length d is already 0.

Extended Reed-Solomon codes are no longer cyclic, but they gain a nice symmetry. If we extend the code defined by RS_G (3) and RS_H (4), we get a new check matrix

$$H_{\text{ext}}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & \dots & 0 \\ 1 & 1 & 1 & 1 & \dots & \dots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \dots & \alpha^{n-k} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \dots & \dots & \alpha^{2n-2k} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{n-1} & \alpha^{n-2} & \alpha^{n-3} & \dots & \dots & \alpha^k \end{bmatrix}$$

I will leave it as an exercise to show that a generator matrix for this code is the transpose of the check matrix $H_{\text{ext}}(n - k)$. If we let $\alpha_{-1} = 0$ and $\alpha_i = \alpha^i$, then

$$G_{ij} = \alpha_j^i \quad \text{where } i = 0, \dots, k - 1; \text{ and } j = -1, 0, 1, \dots, n - 1$$

We may also think of the code as

$$RS_{\text{ext}}(k) = \{(f(\alpha_{-1}), f(\alpha_0), f(\alpha_1), f(\alpha_2), \dots, f(\alpha_{n-1})) : f \in \mathbb{F}_q[x] \text{ and } \deg f < k\}$$

The dual code has the same definition, but with $\deg f < n - k$.

Shortened and punctured Reed-Solomon codes

For a linear code $C \subset \mathbb{F}^n$, the *shortening* of C at the last component is

$$\{(c_1, \dots, c_{n-1}) : (c_1, \dots, c_{n-1}, 0) \in C\}$$

The *puncturing* of C at the last component is

$$\{(c_1, \dots, c_{n-1}) : (c_1, \dots, c_{n-1}, c_n) \in C\}$$

Let \mathbb{F}^{n-1} be the $n - 1$ dimensional subspace of \mathbb{F}^n defined by $x_n = 0$. The shortening of C is the intersection of C with \mathbb{F}^{n-1} and the puncturing of C is the projection of C onto \mathbb{F}^{n-1} .

More generally, given any subset $I \subset \{1, \dots, n\}$, it is clear that we can puncture or shorten relative to the subspace H of \mathbb{F}^n defined by $x_i = 0$ for all $i \in I$.

In general the dimension of an $[n, k, d]$ code shortened in m coordinates is $k - m$, since we are imposing m additional linear conditions on the original code. Of course, it is possible for it to be larger if the conditions are not independent. The minimum distance is the same if there is a codeword of minimum weight whose support does not intersect I . Otherwise the minimum distance will increase.

A punctured $[n, k, d]$ code will generally have dimension k but the dimension may be larger. The minimum distance will generally decrease, because a minimum weight codeword whose support intersects I will puncture to something of strictly lower weight.

Reed-Solomon codes satisfy the general rule. The parameters of a shortened RS code are $[n - m, k - m, n - k + 1]$. A shortened RS code is no longer cyclic (in general). Encoding and decoding can be achieved with little effort from the algorithms we have already developed.

The parameters of a punctured RS code are $[n - m, k, n - m + 1]$.

Exercises 10.1.

1) How would you systematically encode an extended RS code? Discuss how to decode an extended RS code.

2) What modification are necessary to encode and decode a shortened RS code?

11 Assignment: Due 3/11

3.3. #1

4.6 #2, 3, 4. Find the cyclic codes for $n = 14, 15, 16$ and for $q = 2, 4, 3$.

4.6 #6

5.1 #1

12 Assignment: Due 3/20

6.10 #1

8.7 #3

9.5 #1

The following questions all deal with decoding a particular example.

6.10 #2,3

7.1 # 1,2

8.7 #4,5

9.5 #2,3