ECE 7670
Lecture 5 – Cyclic codes

Objective: To become acquainted with the basic concepts of cyclic codes and some aspects of encoder implementations for them.

Reading:

- Chapter 5.

1 Cyclic codes

Definition 1 An (n, k) linear block code C is said to be **cyclic** if for every code word $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ in C, there is also a code word $\mathbf{c}' = (c_{n-1}, c_0, \ldots, c_{n-2})$ that is also in C. ($\mathbf{c}'$ is a cyclic shift of $\mathbf{c}$.) □

It will be convenient to represent our codewords as *polynomials*. The codeword

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$$

is represented by the polynomial

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

using the obvious one-to-one correspondence. A cyclic shift can therefore be represented as follows. Observe that

$$xc(x) = c_0 x + c_1 x + \cdots + c_{n-1} x^n.$$

If we not take this product modulo $x^n - 1$ we get

$$xc(x) \pmod{x^n - 1} = c_{n-1} + c_0 x + \cdots + x_{n-2} x^{n-1}.$$

So multiplication by x in the ring $GF(q)[x]/(x^n - 1)$ corresponds to a cyclic shift. Furthermore, any power of x times a codeword yields a codeword (apply the definition recursively), so that, for example,

$$\begin{aligned}
(c_{n-1}, c_0, c_1, \ldots, c_{n-2}) &\leftrightarrow xc(x) \\
(c_{n-2}, c_{n-1}, c_0, \ldots, c_{n-3}) &\leftrightarrow x^2 c(x) \\
&\vdots \\
(c_1, c_2, \ldots, c_{n-1}, c_0) &\leftrightarrow x^{n-1} c(x)
\end{aligned}$$

where the arithmetic is done in the ring $GF(q)[x]/(x^n - 1)$. Now observe that if we take an polynomial $a(x) \in GF(q)[x]$ of the form

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

then

$$c(x)a(a)$$

is simply a linear combination of cyclic shifts of $c(x)$ and hence, must also be a codeword. Hence: **a cyclic code is an ideal** in $GF(q)[x]/(x^n - 1)$. Because of what we know about ideals in $GF(q)[x]/(x^n - 1)$ we can immediately make some observations about cyclic codes:

- A cyclic code has a generator polynomial $g(x)$, which is the generator of the ideal. Let the degree of g be r, where $r < n$.

- Every code polynomial in the code can be expressed as a multiple of the generator.
$$c(x) = m(x)g(x),$$
where $m(x)$ is the *message polynomial*. The degree of m is less than $n - r$.
- The generator is a factor of $x^n - 1$ in $GF(q)[x]$.

Example 1 We will consider cyclic codes of length 15 with binary coefficients. We need to find the factors of $x^n - 1$ in some field. Observe that

$$15|2^4 - 1,$$

so we are dealing in the field $GF(16)$. The conjugacy classes in $GF(16)$ are

$$\begin{aligned}
\{1\} &\leftrightarrow x + 1 \\
\{\alpha, \alpha^2, \alpha^4, \alpha^8\} &\leftrightarrow 1 + x + x^4 \\
\{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\} &\leftrightarrow 1 + x + x^2 + x^3 + x^4 \\
\{\alpha^5, \alpha^{10}\} &\leftrightarrow 1 + x + x^2 \\
\{\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}\} &\leftrightarrow 1 + x^3 + x^4
\end{aligned}$$

Thus

$$x^{15} - 1 = (x + 1)(1 + x + x^4)(1 + x + x^2 + x^3 + x^4)(1 + x + x^2)(1 + x^3 + x^4)$$

So: degrees 1,2,4,4,4. If we want a generator of, say, degree 9, we could take

$$g(x) = (x + 1)(1 + x + x^4)(1 + x + x^2 + x^3 + x^4)$$

If we want a generator of degree 5 we could take

$$g(x) = (x + 1)(1 + x + x^4)$$

or

$$g(x) = (x + 1)(1 + x + x^2 + x^3 + x^4)$$

In fact, in this case, we can get generator polynomials of any degree from 1 to 15. So we have codes

$$(15, 0)(15, 1), \ldots, (15, 15)$$

□

A message sequence $(m_0, m_1, \ldots, m_{k-1})$ (where $k = n - r$) corresponds to a message polynomial

$$m(x) = m_0 + \cdots + m_{k-1}x^{k-1}.$$

Then the message polynomial corresponding to m is

$$c_m(x) = m(x)g(x).$$

We can write this as

$$c_m(x) = [m_0, m_1, \ldots, m_{k-1}] \begin{bmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{k-1}g(x) \end{bmatrix}$$

Taking the next step, we can go back to a matrix representation,

$$\mathbf{c}_m = \begin{bmatrix} m_0, m_1, \ldots, m_{k-1} \end{bmatrix} \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & & & \\ & g_0 & g_1 & \cdots & g_r & & & & \\ & & g_0 & g_1 & \cdots & g_r & & & \\ & & & \ddots & \ddots & & \ddots & & \\ & & & & g_0 & g_1 & \cdots & g_r & \\ & & & & & g_0 & g_1 & \cdots & g_r \end{bmatrix} = \mathbf{m}G$$

So we have a linear code, and can write the generator matrix corresponding to it. Note: G is $k \times n$.

Let $h(x)$ be **parity check polynomial**, that is a polynomial such that

$$x^n - 1 = g(x)h(x).$$

Since codewords are multiples of $g(x)$, then for a codeword,

$$c(x)h(x) = m(x)g(x)h(x) = m(x)(x^n - 1) \equiv 0 \pmod{x^n - 1}.$$

We let

$$s(x) = c(x)h(x) \pmod{x^n - 1}.$$

be the **syndrome polynomial**. If $s(x)$ is identically zero, then $c(x)$ is a codeword. Now let's put this in matrix form.

$$s(x) = c(x)h(x) = \sum_{i=0}^{n-1} c_i x^i \sum_{j=0}^{n-1} h_j x^j \pmod{x^n - 1}.$$

Performing the multiplication,

$$s_k = \sum_{i=0}^{n-1} c_i h_{((k-i))_n} \qquad k = 0, 1, \ldots, n-1.$$

Writing the last $n - k$ of these out, we have

$$[s_k, s_{k+1}, \ldots, s_{n-1}] = [c_0, c_1, \ldots, c_{n-1}] \begin{bmatrix} h_k & h_{k-1} & \cdots & h_1 & h_0 & & & & & \\ & h_k & h_{k-1} & \cdots & h_1 & h_0 & & & & \\ & & \ddots & \ddots & & & & & & \\ & & & & h_k & h_{k-1} & \cdots & h_1 & h_0 & \\ & & & & & h_k & h_{k-1} & \cdots & h_1 & h_0 \end{bmatrix}^T = \mathbf{c}H^T.$$

Note: H is $n - k \times n$

Example 2 Some cyclic codes of length 7. In $GF(8)$ we have elements of order 7. The minimal polynomials are

$$x + 1 \qquad x^3 + x + 1 \qquad x^3 + x^2 + 1$$

We can choose any combination of these factors as factors of $g(x)$. Let us take

$$g(x) = (x^3 + x + 1)(x + 1) = x^4 + x^3 + x^2 + 1$$

Then $h(x) = x^3 + x^2 + 1$. Here is the list of codewords

$m(x)g(x)$	code polynomial	code word
$0g(x)$	0	0000000
$1g(x)$	$1 + x^2 + x^3 + x^4$	1011100
$xg(x)$	$x + x^3 + x^4 + x^5$	0101110
$x^2g(x)$	$x^2 + x^4 + x^5 + x^6$	0010111
$(x^2 + 1)g(x)$	$1 + x^3 + x^5 + x^6$	1001011
$(x^2 + x + 1)g(x)$	$1 + x + x^4 + x^6$	1100101
$(x + 1)g(x)$	$1 + x + x^2 + x^5$	1110010
$(x^2 + x)g(x)$	$x + x^2 + x^3 + x^6$	0111001

The corresponding generator and parity check matrix are

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

□

The codes to this point have not been systematic, but are easy to produce. We will now do some tricks to make the code be systematic. We take the message block

$$\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \leftrightarrow m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1}.$$

Now take the code polynomial and shift it:

$$x^{n-k} m(x) = m_0 x^{n-k} + m_1 x^{n-k+1} + \cdots + m_{k-1} x^{n-1}.$$

Observe that the code sequence corresponding to this is

$$(0, 0, \ldots, 0, m_0, m_1, \ldots, m_{k-1})$$

Now we employ the division algorithm (this is the trick) to write

$$x^{n-k} m(x) = q(x) g(x) + d(x),$$

where $d(x)$ has degree less than $n - k$. That is, $d(x)$ has the code sequence

$$(d_0, d_1, \ldots, d_{n-k-1}, 0, 0, \ldots, 0)$$

Now form

$$x^{n-k}(x) - d(x) = q(x) g(x).$$

Since the LHS is a multiple of $g(x)$, it is a codeword. It has the form

$$(-d_0, -d_1, \ldots, -d_{n-k-1}, m_0, m_1, \ldots, m_{k-1})$$

We have thus obtained a codeword in which the message data appears explicitly. We have obtained by this trick a systematic code.

Example 3 We will demonstrate systematic coding in the $(7, 3)$ code from before. Let $m(x) = 1 + x^2$ from (1,0,1).

1. Compute $x^{n-k} m(x) = x^4 m(x) = x^6 + x^4$.
2. Employ the division algorithm:

$$x^6 + x^4 = (x^2 + x + 1)(x^4 + x^3 + x^2 + 1) + (x + 1).$$

3. Then our code polynomial is

$$c(x) = x^{n-k} m(x) - d(x) = (1 + x) + (x^6 + x^4) \leftrightarrow (1, 1, 0, 0, 1, 0, 1).$$

We can still explicitly see the message part. □

2 Some hardware considerations

One of the justifications for using cyclic codes, and using the polynomial representation in general, is that there are some interesting and efficient hardware configurations for performing the encoding operation. This has been of particularly historical importance.

2.0.1 Galois field operations

If we represent elements in a Galois field using the m-tuple representation, addition is easy, element-by element, using exclusive-or gates to perform the addition.

Multiplication by a *fixed* Galois field element is also fairly straightforward. Let us take, as an example, multiplication in $GF(2^4)$. Let β be in $GF(2^4)$, and represent it as

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3.$$

That is, β has the m-tuple representation (b_0, b_1, b_2, b_3). Let g be a *fixed* element in $GF(2^4)$, and suppose for this example, that $g = 1 + \alpha$. Let $\chi = \beta g$. Then

$$\begin{aligned}
\chi &= c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3 \\
&= (b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3)(1 + \alpha) \\
&= b_0 + (b_0 + b_1)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3 + b_3\alpha^4 \\
&= (b_0 + b_3) + (b_0 + b_1 + b_3)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3.
\end{aligned}$$

A hardware implementation is straightforward. (Draw a diagram.)

2.0.2 Polynomial multiplication

Now to compute $m(x)g(x)$. This is simply a feed-forward (FIR) filter. The message bits are fed in in descending order: $m_{k-1}, m_{k-2}, \ldots, m_0$:

As a specific example, we have, for $g(x) = 1 + x^2 + x^3 + x^4$ the following

Consider a specific example: $m(x) = m_0 + m_1x + m_2x^2$ and $g(x) = g_0 + g_1x + g_2x^2$. What about further shifts?

For the systematic encoding, we have three steps:

1. Multiply by x^{n-k}. This is accomplished by feeding into the middle of the shift register:

2. Divide by $g(x)$. Polynomial division is obtained using LFSR. To form $a(x)/g(x)$, we recognize (from signal processing experience) that division is accomplished using a feedback connection.

   The inputs are fed in in decreasing index order: $a_{n-1}, a_{n-2}, \ldots, a_0$. After the last symbol has been fed in, the LFSR contains the remainder of $a(x)/g(x)$.

3. Set $c(x) = x^{n-k}m(x) - d(x)$. Here we simply employ some switching circuitry:

Start in X: pass in message, which appears at output and goes into divider. Then switch to Y and read out the parity bits.

3 Syndrome decoding

We now examine the question of decoding cyclic codes. Recall that for any linear code, we can form a standard array; or we can use the reduced standard array using syndromes. We will be able to exploit the cyclic structure of the codes to decrease the memory requirements.

Consider the systematically-encoded codeword

$$\mathbf{c} = (-d_0, -d_1, \ldots, -d_{n-k-1}, m_0, m_1, \ldots, m_{k-1}) = (-\mathbf{d}, \mathbf{m})$$

We can perform error *detection* as follows:

1. Estimate a message based on the systematic message part. Call this $\mathbf{m}'$.
2. We then encode $\mathbf{m}'$. Compute the parity bits from this to the received parity bits. If they don't match, then an error is detected.

We can also use shift-register techniques. Recall that if we take $r(x)h(x)$ modulo $x^n - 1$, we get a syndrome polynomial. But since

$$g(x)h(x) = x^n - 1$$

we can use the division algorithm, dividing by $g(x)$

$$r(x) = a(x)g(x) + s(x).$$

But the division we can compute as we saw before.

We have the following useful result about cyclic codes and syndromes.

theorem 1 *Let $s(x)$ be the syndrome corresponding to $r(x)$. Let $r^{(1)}(x)$ be the polynomial obtained by cyclicly right-shifting $r(x)$. Then the remainder obtained when dividing $xs(x)$ by $g(x)$ is the syndrome denoted $s^{(1)}$ corresponding to $r^{(1)}(x)$. In other words, shifts of r (mod $x^n - 1$) correspond to shifts of s (modulo g).*

Proof Write

$$r^{(1)}(x) = xr(x) - r_{n-1}(x^n - 1).$$

(Show how this works.) Let us write

$$r(x) = a(x)g(x) + s(x).$$

Then

$$r^{(1)}(x) = x[a(x)g(x) + sb(x)] - r_{n-1}[g(x)h(x)].$$

Let us write this as

$$r^{(1)}(x) = b(x)g(x) + d(x).$$

We note that $d(x)$ must be the syndrome of $r^{(1)}(x)$. We also note from the previous result, solving for $xs(x)$,

$$xs(x) = [b(x)g(x)+d(x)]-xa(x)g(x)+r_{n-1}g(x)h(x) = (b(x)-xa(x)+r_{n-1}h(x))g(x)+d(x).$$

Thus $d(x)$ is the remainder of $xs(s)$ divided by $g(x)$, establishing the theorem. □

We conclude we can reduce the size of the syndrome lookup table. We need to only store one syndrome $\mathbf{s}$ for an error $\mathbf{e}$ and all cyclic shifts of $\mathbf{e}$. And we can compute the shifts necessary using the same circuit that computes the syndrome in the first place.

Example 4

error	syndrome	syndrome poly
0000000	000	0
1000000	100	1
0100000	010	x
0010000	001	x^2
0001000	110	$1 + x$
0000100	011	$x + x^2$
0000010	111	$1 + x + x^2$
0000001	101	$1 + x^2$

Comment on the pattern. So all we have to do in this case is look for is any one of the syndrome patterns (instead of many of them). We choose to look for 101,

since if we find the error in that location, we can shift correct it on the end.

□

4 CRC codes

CRC codes are very commonly used for ARQ (automatic repeat request). Every modem and network protocol uses ARQ, reverse error correction, even where forward error correction may not be. They provide powerful error detection capability, and are easily implemented. They are appropriate to talk about here, because they are basically cyclic codes, with syndrome computation computed using LFSR.

CRC codes are simply binary cyclic codes with particular polynomials, chosen to obtain good error detection capability.

We will explain the idea using notation from *Data and Computer Communications* by W. Stallings (5th Ed., Prentice-Hall). Let $M(x)$ be a message polynomial of degree $k-1$, and let $R(X)$ be a parity check polynomial of degree $n-1$. Then form

$$T(x) = x^n M(x) + R(x).$$

Then we have a systematic representation of $M(x)$ in T. This is the encoding operation. The decoder takes the received signal and divides by some polynomial $P(x)$. We want to choose $R(x)$ so that the remainder when $T(x)$ is divided by $P(x)$

is exactly zero. We obtain $R(x)$ as follows: Observe that we can write

$$\frac{x^n M(x)}{P(x)} = Q(x) + \frac{r(x)}{P(x)}$$

Then if we set $R(x) = r(x)$ (the remainder) we get the zero remainder that we want. That is, we encode by forming

$$T(x) = x^n M(x) + R(x).$$

Then we decode by dividing:

$$\frac{T(x)}{P(x)} = \frac{x^n M(x) + R(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)} + \frac{R(x)}{P(x)} = Q(x).$$

(all operations in $GF(2)$).

Example 5 Let $M = 1010001101 \leftrightarrow M(x) = x^9 + x^7 + x^3 + x^2 + 1$ and $P = 110101 \leftrightarrow P(x) = x^5 + x^4 + x^2 + 1$.

1. Message is shifted by 2^5, yielding 10100011010000.
2. Product is divided by P:
3. The remainder $R = 01110$ is added to give $T = 101000110101110$.

4. At the receiver, divide again by P:

If there is no remainder, it is assumed that there have been no errors.

□

A couple of common polynomials:

$$\text{CRC-16} x^{16} + x^{14} + x^2 + 1$$
$$\text{CRC-CCITT} x^{16} + x^{12} + x^5 + 1$$

How well do they work? We first observe that an error is undetectable only if it is divisible by $P(x)$. A summary of some performance results follows for detectable errors.

- All single-bit errors.
- All double-bit errors if weight$(P(x)) \geq 3$.
- Any odd number of errors if $x + 1 | P(x)$.
- Any burst error for which the length of the burst is less than the length of $P(x)$.
- "Most" larger burst errors.

The book provides some specifics: CRC-16 or CRC-CCITT detects the following:

1. All single bursts of length 16 or less.
2. 99.997% of all error bursts of length 17.
3. 99.9985% of all error bursts of length > 17.