

A Short Introduction to Cyclic Codes

Henry D. Pfister

November 2nd, 2017

1 Cyclic Codes

1.1 Basic Properties

In this section, we consider a subset of linear codes which have even more algebraic structure. A cyclic (or circular) shift of a vector is another vector with the same elements in the same order, but starting from a different index. For example, the right circular shift of $(x_0, x_1, \dots, x_{n-1})$ by 1 position gives the vector $(x_{n-1}, x_0, x_1, \dots, x_{n-2})$.

Definition 1. A **cyclic code** is a linear code where any cyclic shift of a codeword is also a codeword.

Example 2. Consider the $(7, 3)$ binary linear code whose 8 codewords are

$$\mathcal{C} = \{0000000, 1011100, 0101110, 1110010, 0010111, 1001011, 0111001, 1100101\}.$$

Since all non-zero codewords are circular shifts of a single codeword, it is a cyclic code. Its cyclic structure can also be exposed by choosing the generator matrix to be

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

and the parity-check matrix to be

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

The benefits of cyclic codes follow largely from a close connection to polynomial arithmetic. This allows the encoding and decoding of cyclic codes to make use of efficient algorithms and hardware for polynomial arithmetic. Consider an (n, k) cyclic code \mathcal{C} and assume each codeword vector $\underline{c} = (c_0, c_1, \dots, c_{n-1})$ and message vector $\underline{m} = (m_0, m_1, \dots, m_{k-1})$ is associated with a polynomial

$$\underline{c} \leftrightarrow c(x) = \sum_{i=0}^{n-1} c_i x^i \quad \underline{m} \leftrightarrow m(x) = \sum_{i=0}^{k-1} m_i x^i.$$

As we will see shortly, every (n, k) cyclic code is uniquely defined by its generator polynomial $g(x)$, which allows all codewords $c(x) = m(x)g(x)$ to be generated by multiplication with some message polynomial $m(x)$. The generator polynomial for the code in the previous example is $g(x) = 1 + x^2 + x^3 + x^4$.

Before proceeding any further, we must define a few mathematical terms associated with polynomial arithmetic. The **degree** of a polynomial $c(x)$, denoted $\deg(c(x))$, is the maximum power of x that appears in the polynomial. For the example $g(x)$, we have $\deg(g(x)) = 4$. In fact, for all (n, k) cyclic codes, we have $\deg(c(x)) \leq n - 1$, $\deg(m(x)) \leq k - 1$, and $\deg(g(x)) = n - k$.

For a polynomial $a(x)$ and a divisor $d(x)$, the **remainder** $r(x)$ and **quotient** $q(x)$ are uniquely defined by $\deg(r(x)) < \deg(d(x))$ and

$$a(x) = q(x)d(x) + r(x).$$

In discrete mathematics, the **modulo polynomial** operation

$$r(x) = a(x) \bmod d(x)$$

is used to compactly represent the remainder $r(x)$ of division by $d(x)$.

Using these operations, the cyclic-shift property of a cyclic code is equivalent to the algebraic statement: if $\underline{c} \in \mathcal{C}$ with $\underline{c} \leftrightarrow c(x)$, then

$$(x_{n-1}, x_0, x_1, \dots, x_{n-2}) = \tilde{c} \leftrightarrow \tilde{c}(x) = xc(x) \bmod x^n - 1$$

and $\tilde{c} \in \mathcal{C}$. Combined with linearity, this implies that the code \mathcal{C} is isomorphic to a principal ideal in the quotient ring $\mathbb{F}_2[x]/\langle x^n - 1 \rangle$. In addition, this ideal is generated by the minimum-degree non-zero element $g(x)$ and one can show that $g(x)$ divides $x^n - 1$. It follows that the ideal can be written as $c(x) = m(x)g(x)$ where $\deg(m(x)) \leq k - 1$.

Proposition 3. *For an (n, k) cyclic code defined by its generator polynomial $g(x)$, the parity-check polynomial $h(x)$ is defined uniquely by $g(x)h(x) = x^n - 1$ and satisfies*

$$h(x)c(x) \bmod x^n - 1 = 0,$$

for all codeword polynomials $c(x)$.

Proof. First, we observe that $h(x)$ can be computed by dividing $x^n - 1$ by $g(x)$. Therefore, one can prove this statement by observing that $c(x) = m(x)g(x)$ for some $m(x)$ and therefore

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

Since $x^n - 1$ divides $h(x)c(x)$ without remainder, the result follows. \square

The parity-check polynomial for the code in the previous example is $h(x) = 1 + x^2 + x^3$.

In general, the generator matrix of an (n, k) cyclic code with generator $g(x) = g_0 + g_1x + g_2x^2 + \cdots + g_{n-k}x^{n-k}$ can be written in the form

$$\underline{G} = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & g_0 & \cdots & g_{n-k-1} & g_{n-k} \end{bmatrix}.$$

Likewise, the parity-check polynomial $h(x) = h_0 + h_1x + h_2x^2 + \cdots + h_kx^k$ gives rise to a parity-check matrix of the form

$$\underline{H} = \begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_k & \cdots & h_1 & h_0 \end{bmatrix}.$$

The encoder mapping $\underline{c} = \underline{m}\underline{G}$ is actually identical to the encoder mapping $c(x) = m(x)g(x)$. If the rows and columns of \underline{G} are numbered from zero, then this can be seen by associating the i th row of \underline{G} with x^i and the j th column of \underline{G} with x^j . Observing that $[\underline{G}]_{ij} = g_{j-i}$ for $i \leq j \leq n-1$ and 0 otherwise, we find that

$$\begin{aligned} c(x) &= \sum_{j=0}^{n-1} x^j c_j = \sum_{j=0}^{n-1} x^j \sum_{i=0}^{k-1} m_i [\underline{G}]_{ij} = \sum_{j=0}^{n-1} x^j \sum_{i=0}^{k-1} m_i g_{j-i} \\ &= \sum_{i=0}^{k-1} m_i x^i \sum_{j=i}^{n-1} x^{j-i} g_{j-i} = \sum_{i=0}^{k-1} m_i x^i g(x) = m(x)g(x). \end{aligned}$$

Example 4. The binary Hamming codes are a family of (n, k) cyclic codes, with $n = 2^r - 1$ and $k = n - r$, that can correct all single errors. The generator polynomials for $r = 3, 4, 5$ are given by

$$\begin{aligned} (7, 4) \quad & g(x) = 1 + x^2 + x^3 \\ (15, 11) \quad & g(x) = 1 + x^3 + x^4 \\ (31, 26) \quad & g(x) = 1 + x^3 + x^5. \end{aligned}$$

1.2 Systematic Encoding

For cyclic codes, the basic encoding process $c(x) = m(x)g(x)$ does not provide systematic encoding. One can overcome this limitation by observing that $c(x) \bmod g(x) = 0$ for all codewords. Using this equation instead, one can choose the codeword to have the format

$$c(x) = p(x) + x^{n-k}m(x),$$

where the message $m(x)$ appears as the last k bits in the codeword and the parity polynomial $p(x) = p_0 + p_1x + \cdots + p_{n-k-1}x^{n-k-1}$ has degree $\deg(p(x)) = n - k - 1$. Applying the modulo

Example 6. For case of $n = 7$, we have $x^7 - 1 = (x^3 + x + 1)(x^3 + x^2 + 1)(x + 1)$. Thus, there are exactly 2^3 ways to choose factors of $x^7 - 1$ and each choice defines a distinct cyclic code. For example, the $(7, 3)$ simplex code is generated by

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

1.4 Cyclic Redundancy Check

A **cyclic redundancy check** (CRC) code calculates its check bits using a process very similar to the systematic encoding of cyclic codes. The main difference is that the message polynomial $m(x)$ is not constrained to be k bits in length and it is not multiplied by x^{n-k} before the modulo operation. Let $g(x)$ be the CRC polynomial and $m(x)$ be the message polynomial, then the CRC encoder generates the check polynomial

$$p(x) = m(x) \mod g(x).$$

Since $\deg(p(x)) < \deg(g(x))$, the resulting check sequence is represented by exactly $\deg(g(x)) - 1$ bits. Many of the CRCs defined in standards (or commonly used in practice) have subtle variations such as

- extra bits are appended to the start or end of the message,
- the shift register state is initialized to a non-zero value,
- the message sequence is XOR'd with a known sequence before division,
- and the bit (or byte) ordering may differ between standards.

In most specifications, the CRC polynomial is not described as a polynomial but instead given as the hexadecimal value of the binary representation of $g(x)$. Moreover, the topmost bit is not included in this value because the length is known and it must be a 1.

Example 7. The standard unix CRC32 polynomial is often listed in hexadecimal as 04C11DB7. Writing this in binary gives

$$0000\ 0100\ 1100\ 0001\ 0001\ 1101\ 1011\ 0111.$$

Counting from the right (i.e., the LSB), we find ones in the positions

$$0, 1, 2, 4, 5, 7, 8, 10, 11, 12, 16, 22, 23, 26.$$

Therefore, it follows that

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

Notice the x^{32} is added based on the known length of the CRC and is not described by the hexadecimal value.

The CRC32 algorithm prepends the data sequence by 32 ones, performs the modulo, and then inverts the resulting parity sequence. The ASCII sequence “123456789” has the hexadecimal representation 31 32 33 34 35 36 37 38 39 and is commonly used to test CRC implementations. For this sequence, the correct CRC result, in hexadecimal, is CBF43926.