Systematic Cyclic Codes

Polynomial multiplication encoding for cyclic linear codes is easy. Unfortunately, the codes obtained are in most cases not *systematic*. Systematic cyclic codes can be obtained through a procedure that is only slightly more complicated than the polynomial multiplication procedure.

Systematic Encoding

Consider an (n, k) cyclic code C with generator polynomial $g(x)$. The k-symbol message block is given by the message polynomial $m(x)$.

Step 1. Multiply the message polynomial $m(x)$ by x^{n-k}.

Step 2. Divide the result of Step 1 by the generator polynomial $g(x)$. Let $d(x)$ be the remainder.

Step 3. Set $c(x) = x^{n-k}m(x) - d(x)$.

This encoding works, as (1) $c(x)$ is a multiple of $g(x)$ and therefore a codeword, (2) the first $n-k$ coefficients of $x^{n-k}m(x)$ are zero, and (3) only the first $n-k$ coefficients of $-d(x)$ are nonzero (the degree of $g(x)$ is $n-k$).

Example: Systematic Encoding (1)

We consider the $(7, 3)$ binary cyclic code with generator polynomial $g(x) = x^4 + x^3 + x^2 + 1$ discussed in a previous example, and encode $101 = 1 + x^2 = m(x)$.

Step 1. $x^{n-k}m(x) = x^4(x^2 + 1) = x^6 + x^4$.

Step 2. $x^6 + x^4 = (x^4 + x^3 + x^2 + 1)(x^2 + x + 1) + (x + 1)$, so $d(x) = x + 1$ (*Carry out the necessary division in the same way as you learnt in elementary school!*).

Step 3. $c(x) = x^6 + x^4 - (x + 1) = 1 + x + x^4 + x^6$, and the transmitted codeword is 1100101.

Example: Systematic Encoding (2)

The systematic generator matrix is obtained by selecting as rows the codewords associated with the messages 100, 010, and 001. The parity check matrix is obtained using the basic result (presented earlier) that $\mathbf{H} = [\mathbf{I}_{n-k} \mid -\mathbf{P}^T]$ with $\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k]$. In the current example, we get

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right], \quad \mathbf{H} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right].$$

Implementations of Cyclic Codes

Data rates are very high in many applications $\Rightarrow$ only very fast decoders and encoders can be used.

Fast circuits are, for example, simple exclusive OR (XOR) gates, switches, and shift registers. For nonbinary encoders and decoders, finite-field adder and multiplier circuits are needed. We now focus on *shift-register* (SR) encoders and decoders for cyclic codes.

Operational Elements in Shift Registers

The symbology used is depicted in [Wic, Fig. 5-1].

half-adder Adds the input values without carry. In the binary case, XOR.

SR cell Flip-flops. In the binary case, one.

fixed multiplier Multiplies the input value with a given value. In the binary case, existence or absence of connection.

In the nonbinary case, we assume that the field is a *binary* extension field: $\mathrm{GF}(p^m)$ with $p = 2$. The circuits are substantially more complicated when $p \neq 2$.

Addition and SRs in Extension Fields

Elements $\alpha, \beta \in \mathrm{GF}(2^m)$ are represented as binary m-tuples $(a_0, a_1, \ldots, a_{m-1})$ and $(b_0, b_1, \ldots, b_{m-1})$, respectively.

Then the addition of α and β gives $(a_0 + b_0, a_1 + b_1, \ldots, a_{m-1} + b_{m-1})$, where $+$ is binary addition. The nonbinary addition circuit is shown in [Wic, Fig. 5-2].

The non-binary shift-register cells are implemented with one flip-flop for each coordinate in the m-tuple; see [Wic, Fig. 5.4].

Multiplication in Extension Fields

As an example, we consider multiplication in $\mathrm{GF}(2^4)$ of an arbitrary value $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ by a *fixed* value $g = 1 + \alpha$, where α is a root of the primitive polynomial $x^4 + x + 1$. Then

$$\begin{aligned}
\beta \cdot g &= (b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3)(1 + \alpha) \\
&= b_0 + (b_0 + b_1)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3 + b_3\alpha^4 \\
&= b_0 + (b_0 + b_1)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3 + b_3(\alpha + 1) \\
&= (b_0 + b_3) + (b_0 + b_1 + b_3)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3.
\end{aligned}$$

The corresponding multiplier circuit is illustrated in [Wic, Fig. 5-3].

Nonsystematic Encoders

With message polynomial $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1}$ and generator polynomial $g(x)$, the codeword polynomial is

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= m_0 g(x) + m_1 x g(x) + \cdots + m_{k-1}x^{k-1}g(x). \end{aligned}$$

The corresponding SR circuit is shown in [Wic, Fig. 5-5].

Systematic Encoders

Step 1. (Multiply $m(x)$ by x^{n-k}.) Easy, shown in [Wic, Fig. 5-8].

Step 2. (Divide the result of Step 1 by $g(x)$, and let $d(x)$ be the remainder.) Polynomial division is carried out through the use of a linear feedback shift register (LFSR) as shown in [Wic, Fig. 5-9], where $a(x)$ is divided by $g(x)$, and $q(x)$ and $d(x)$ are the quotient and remainder, respectively.

Step 3. (Set $c(x) = x^{n-k}m(x) - d(x)$.) Achieved by combining the two SR circuits for the previous steps, as shown in [Wic, Fig. 5-12].

An alternative encoder for cyclic codes, not considered here, is presented in [Wic, Fig. 5-13].

Error Detection for Systematic Codes

The transmitted codeword of a systematic cyclic code has the form

$$\mathbf{c} = (c_0, c_1, \ldots, c_n) = (\underbrace{-d_0, -d_1, \ldots, -d_{n-k-1}}_{\text{remainder block}}, \underbrace{m_0, m_1, \ldots, m_{k-1}}_{\text{message block}}).$$

Error detection is performed on a received word $\mathbf{r}$ as follows.

1. Denote the values in the message and parity positions of the received word $\mathbf{r}$ by $\mathbf{m}$ and $\mathbf{d}$, respectively.
2. Encode $\mathbf{m}$ using an encoder identical to that used by the transmitter, and denote the remainder block obtained in this way by $\mathbf{d}'$.
3. Compare $\mathbf{d}$ with $\mathbf{d}'$. If they are different, then the received word contains errors.

Syndrome Computation for Systematic Codes

Denote the received word by $\mathbf{r}$ with $\mathbf{m}$ and $\mathbf{d}$ in the message and parity positions, respectively. Let $\mathbf{d}'$ be a valid parity block of message $\mathbf{m}$ (cf. previous slide), and denote this valid word by $\mathbf{r}'$.

$$\begin{aligned} \mathbf{s} &= \mathbf{r}\mathbf{H}^T \\ &= (\mathbf{r} - \mathbf{r}')\mathbf{H}^T \ (\text{as } \mathbf{r}'\mathbf{H}^T = \mathbf{0}) \\ &= (\underbrace{d_0 - d_0', d_1 - d_1', \ldots, d_{n-k-1} - d_{n-k-1}'}_{\mathbf{d}-\mathbf{d}'}, 0, 0, \ldots, 0)\mathbf{H}^T \\ &= \mathbf{d} - \mathbf{d}', \end{aligned}$$

since the parity check matrix has the form $\mathbf{H} = [\mathbf{I}_{n-k} \mid -\mathbf{P}^T]$.

Syndromes for nonsystematic codes can also be computed through the use of shift registers.

Error-Correction Approaches

Error correction has earlier been discussed for general linear codes.

- ▷ A *standard array* has q^n entries.
- ▷ A *syndrome table* has q^{n-k} entries.
- ▷ We shall see that the number of entries of a syndrome table for cyclic linear codes can be reduced to approximatively q^{n-k}/n.
- ▷ With more (algebraic) structure of the codes, even more powerful decoding is possible (to be discussed in forthcoming lectures).

Syndrome Decoding for Cyclic Codes

Theorem 5-3. Let $s(x)$ be the syndrome polynomial corresponding to a received polynomial $r(x)$. Let $r_i(x)$ be the polynomial obtained by cyclically shifting the coefficients of $r(x)$ i steps to the right. Then the remainder obtained when dividing $xs(x)$ by $g(x)$ is the syndrome $s_1(x)$ corresponding to $r_1(x)$.

Having computed the syndrome $\mathbf{s}$ with an SR division circuit, we get $s_i(x)$ after the input of i 0s into the circuit! We then need only store one syndrome $\mathbf{s}$ for an error pattern $\mathbf{e}$ and all cyclic shifts of $\mathbf{e}$.

Decoding Algorithm for Cyclic Codes

1. Let $i := 0$. Compute the syndrome $\mathbf{s}$ for a received vector $\mathbf{r}$.
2. If $\mathbf{s}$ is in the syndrome look-up table, goto Step 6.
3. Let $i := i + 1$. Enter a 0 into the SR input, computing $\mathbf{s}_i$.
4. If $\mathbf{s}_i$ is not in the syndrome look-up table, goto Step 3.
5. Let $\mathbf{e}_i$ be the error pattern corresponding to the syndrome $\mathbf{s}_i$. Determine $\mathbf{e}$ by cyclically shifting $\mathbf{e}_i$ i times to the left.
6. Let $\mathbf{c} := \mathbf{r} - \mathbf{e}$. Output $\mathbf{c}$.

Example: Error Correction of (7,4) Cyclic Code

Consider the (7,4) binary cyclic code generated by $g(x) = x^3 + x + 1$, with parity check polynomial $h(x) = (x^7 + 1)/g(x) = x^4 + x^2 + x + 1$, and with parity check matrix

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

This is a one-error-correcting Hamming code, so all correctable error patterns are cyclic shifts of 0000001. An SR error-correction circuit for this code is displayed in [Wic, Fig. 5-14].

Error Detection in Practice

The most frequently used error control techniques in the history of computers and communication networks are:

one-bit parity check Very simple, but yet important.

CRC codes Shortened cyclic codes that have extremely simple and fast encoder and decoder implementations.

Properties of CRC Codes

- **Cyclic redundancy check (CRC)** codes are shortened cyclic codes obtained by deleting the j rightmost coordinates in the codewords.
- CRC codes are generally not cyclic.
- CRC codes can have the same SR encoders and decoders as the original cyclic code.
- CRC codes have error detection and correction capabilities that are at least as good as those of the original cyclic code.
- CRC codes have good burst-error detection capabilities.

Some Generator Polynomials

CRC-4	$g_4(x) = x^4 + x^3 + x^2 + x + 1$
CRC-12	$g_{12}(x) = (x^{11} + x^2 + 1)(x + 1)$
CRC-ANSI	$g_A = (x^{15} + x + 1)(x + 1)$
CRC-CCITT	$g_C = (x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1) \cdot (x + 1)$

Example. The polynomial $g_{12}(x)$ divides $x^{2047} - 1$ but no polynomial $x^m - 1$ with smaller degree, so it defines a cyclic code of length 2047 and dimension $2047 - 12 = 2035$. So, CRC-12 encodes up to 2035 message bits, generating 12 bits of redundancy.

Error Detection Performance Analysis

The error detection performance of codes depends of the type of errors. In performance analysis, the following three situations are most often considered.

1. *Total corruption of words.*
2. *Burst errors.* These are errors that occur over several *consecutive* transmitted symbols.
3. *The binary symmetric channel.*

Total Corruption of Words

When an (n, k) code is used, total corruption leads to a decoder error with probability

$$\frac{q^k}{q^n} = q^{k-n}.$$

Note that this probability is solely a function of the number of redundant symbols in the transmitted codewords.

Example. With CRC-12, an error is detected with probability $1 - 2^{-12} \approx 0.999756$ in case of total corruption.

Burst-Error Detection

A burst-error pattern of length b starts and ends with nonzero symbols; the intervening symbols may be take on any value, including zero.

Theorems 5-4, 5-5, and 5-6. A q-ary cyclic or shortened cyclic codes with generator polynomial $g(x)$ of degree r can detect *all* burst error patterns of length r or less; the fraction $1 - q^{1-r}/(q-1)$ of burst error patterns of length $r + 1$; and the fraction $1 - q^{-r}$ of burst error patterns of length greater than $r + 1$.

Example. With CRC-12, all bursts of length at most 12, 99.95% of bursts of length 13, and 99.976% of longer bursts are detected.

The Binary Symmetric Channel

An exact determination of the performance of a CRC code over the binary symmetric channel requires knowledge of the weight distribution of the code.