Error Control Strategies

- Forward Error Correction (FEC)
- Automatic Repeat Request (ARQ)
Forward Error Correction (FEC)

In a one-way communication system: The transmission or recording is strictly in one direction, from transmitter to receiver. Error control strategy must be FEC; that is, they employ error-correcting codes that automatically correct errors detected at the receiver.

Examples:
- digital storage systems, in which the information recorded can be replayed weeks or even months after it is recorded, and
- deep-space communication systems.

Most of the communication systems in use today employ some form of FEC, even if the channel is not strictly one-way.
**Automatic Repeat Request (ARQ)**

In most communication systems, the information can be sent in both directions, and the transmitter also acts as a receiver (transceiver), and vice-versa.

**Examples:** data networks, satellite communications, etc.

Error control strategies for a two-way system can include error detection and retransmission, called automatic repeat request (ARQ).

In an ARQ system, when errors are detected at the receiver, a request is sent for the transmitter to repeat the message, and repeat requests continue to be sent until the message is correctly received.
Remarks (ARQ Versus FEC)

✓ The major advantage of ARQ versus FEC is that error detection requires much simpler decoding equipment than error correcting.

✓ Also, ARQ is adaptive in the sense that information is retransmitted only when errors occurs.

✓ In contrast, when the channel error is high, retransmissions must be sent too frequently, and the **SYSTEM THROUGHPUT** – the rate at which newly generated messages are correctly received- is lowered by ARQ.

In this situation, a **HYBRID combination** of FEC for the most frequent error patterns along with error detection and retransmission for the less likely error patterns is more efficient than ARQ alone.
The performance of a coded communication system is in general measure by its probability of decoding error (called the error probability) and its coding gain over the uncoded system that transmit information at the same rate.

There are two types of error probabilities, probability of word (or block) error and probability of bit error. The probability of block error is defined as the probability that a decoded word (or block) at the output of the decoder is in error. This error probability is often called the word-error rate (WER) or block-error rate (BLER). The probability of bit-error rate, also called the bit error rate (BER), is defined as the probability that a decoded information bit at the output of the decoder is in error.

A coded communication system should be designed to keep these two error probabilities as low as possible under certain systems constraints, such as power, bandwidth and decoding complexity.
Performance Measures (cont’d)

- The error probability of a coded communication system is commonly expressed in terms of the ratio of energy-per information bit, $E_b$, to the one-sided power spectral density (PSD) $N_0$ of the channel noise.

Example 1

Consider a coded communication system using an (23, 12) binary Golay code for error control. Each codeword consists of 23 code digits, of which 12 are of information. Therefore, there are 11 redundant bits, and the code rate is $R=12/23=0.5217$.

Suppose that BPSK modulation with coherent detection is used and the channel is additive white Gaussian noise (AWGN), with one-side PSD $N_0$. Let $E_b/N_0$ at the input of the receiver be the **signal-to-noise ratio (SNR)**, which is usually expressed in dB.

The bit-error performance of the (23,12) Golay code with both hard- and soft-decision decoding versus SNR is given, along with the performance of the uncoded system.
Example 1 (cont’d)

**FIGURE 1.** Bit-error performance of a coded communication system with the (23, 12) Golay code.
Example 1 (cont’d)

Remarks (from Fig. 1):

✓ The coded system, with either hard- or soft-decision decoding, provides a lower bit-error probability than the uncoded system for the same SNR, when the SNR is above a certain threshold.

Hard-decision (see Fig. 1), this threshold is 3.7dB.

For SNR=7dB, the BER of the uncoded system is $8 \times 10^{-4}$, whereas the coded system (hard-decision) achieves a BER of $2.9 \times 10^{-5}$. This is a significant improvement in performance.

For SNR=5dB this improvement in performance is small: $2.1 \times 10^{-3}$ compared to $6.5 \times 10^{-3}$.
However, with soft-decision decoding, the coded system achieves a BER of $7 \times 10^{-5}$. 

6876: Communication Networks
The other performance measure is the **CODING GAIN**. Coding gain is defined as the reduction in SNR required to achieve a specific error probability (BER or WER) for a coded communication system compared to an uncoded system.

**Example 1 (cont’d)**

For a **BER=10^{-5}**, the Golay-coded system with hard-decision decoding has a coding gain of 2.15dB over the uncoded system, whereas with soft-decision decoding, a coding gain of more than 4dB is achieved. This result shows that **soft-decision decoding** of the Golay code achieves 1.85dB **additional coding gain compared to hard-decision decoding** at a BER of 10^{-5}. This additional coding gain is achieved at the expense of **higher decoding complexity**.

**Coding gain is important in communication applications,** where every dB of improved performance results in savings in overall system cost.
Example 1 (cont’d)

Remarks:

✓ At sufficient low SNR, the coding gain actually becomes negative. This threshold phenomenon is common to all coding schemes. There always exists an SNR below which the code loses its effectiveness and actually makes the situation worse. This SNR is called the CODING THRESHOLD.

It is important to keep this threshold low and to maintain a coded communication system operating at an SNR well above its coding threshold.

Another quantity that is sometimes used as a performance measure is the ASYMPTOTIC CODING GAIN (the coding gain for large SNR).
Shannon’s Limit as A Function of Code Rate:

✓ In designing a coding system for error control, it is desired to minimize the SNR required to achieve a specific error rate. This is equivalent to maximizing the coded gain of the coded system compared to an uncoded system using the same modulation format.

✓ A theoretical limit on the minimum SNR required for a coded system with code rate $R$ to achieve error-free communication (or an arbitrarily small error probability) can be derived based on Shannon’s noisy coding theorem. This theoretical limit, often called the Shannon limit, simply says that for a coded system with code rate $R$, error-free communication is achieved only if the SNR exceeds this limit. As long as SNR exceeds this limit, Shannon’s theorem guarantees the existence of a (perhaps very complex) coded system capable of achieving error-free communication.

✓ For transmission over a binary-input, continuous-output AWGN with BPSK signaling, the Shannon’s limit, in terms of SNR as a function of the code rate does not have a close form; however, it can be evaluated numerically.
Shannon’s Limit as A Function of Code Rate (cont’d):

![Graph showing Shannon limit as a function of code rate R.](image)

**Figure 2**: Shannon limit as a function of code rate $R$. 
Shannon’s Limit as A Function of Code Rate (cont’d):

- Uncoded
- Coded $R = 1/2$

0.188 dB
Shannon’s Limit as A Function of Code Rate (cont’d):

✓ From Fig. 2 (Shannon limit as a function of the code rate for BPSK signaling on a continuous-output AWGN channel), one can see that the **minimum required SNR to achieve error free communication with a coded system with rate** $R = 1/2$, is **0.188dB**.

✓ The Shannon limit can be used as a yardstick to measure the **maximum achievable coding gain for a coded system with a given rate** $R$ **over an uncoded system with the same modulation format**.

From Fig 3. one can notice that to achieve $\text{BER} = 10^{-5}$, an **uncoded** BPSK system requires an SNR of **9.65dB**. On the other hand, for a coded system with code rate $R = 1/2$, the Shannon limit is **0.188dB** (from Fig. 2). Therefore, the **maximum potential coding gain** for a coded system with code rate $R = 1/2$ is **9.462dB**.

For example, a rate $R = 1/2$ convolutional code with memory order 6, achieves $\text{BER} = 10^{-5}$ with **SNR=4.15dB**, and achieves a code gain of **5.35dB** compared to the uncoded system (Fig. 3). However, **it is 3.962dB away from the Shannon’s limit**. This gap can be reduced by using a longer and more powerful code.
Summary: Basic Concepts in Error Control

**Error detection**
- Transmitter
- Data
- ARQ (automatic repeat request)
- Receiver
- Error detected
- Discard

**Error correction**
- Transmitter
- Data
- Receiver
- Error detected
- Correct (FEC)
**ERROR CONTROL STRATEGIES**

**History (FEC):**

- **Hamming codes (1948)** Bell Lab
- **Golay codes (1949)** Voyager, (Jupiter’79, Saturn’81)
- **Reed-Muller (‘6x), Reed-Solomon codes**
- **Convolutional codes**
- **Turbo codes (1993)**
- **Low-density parity check codes**
Random Error Channels: are memoryless channels; the noise affects each transmitted symbol independently. Example: deep space and satellite channels, most line-of-sight transmission.

Burst Error Channels: are channels with memory. Example: fading channels (the channel is in a “bad state” when a deep fade occurs, which is caused by multipath transmission) and magnetic recordings subject to dropouts caused by surface defects and dust particles.

Compound Channels: both types of errors are encountered.
Shannon’s Information Theory and Coding

Source coding

Channel coding

channel

Channel decoding

Source decoding

Compression

Error Detection and Correction

Channel Capacity

Sent messages

Received messages

0110101001110...

0110101001110...

6876: Communication Networks
Error Detection and Error Correction Capability

A code can be characterized in terms of its amount of error detection capability and error correction capability. The **error detection capability** is the ability of the decoder to tell if an error has been made in transmission. The **error correction capability** is the ability of the decoder to tell which bits are in error.

**Binary Code, \( M=\{0,1\} \)**

\[
\overline{m} = (m_0, \ldots, m_{k-1})
\]

**Set of Cod Words, \( C \)**

\[
\overline{c} = (c_0, \ldots, c_{n-1}), \quad n > k
\]

- **Message**: block of \( k \) bits
- **Code Word**: block of \( n \) bits

\( G \) is the encoding rule

- Only \( 2^k \) out of \( 2^n \) are used as code words.
ERROR DETECTION and CORRECTION

Error Detection and Error Correction Capability (cont’d)

• **Message**: block of \( k \) bits

• **Assumptions**:
  - independent bits
  - each message is equally probable

• \( 2^k \) equally likely messages, of \( k \) bits each

• **Code Word**: block of \( n \) bits

  • Only \( 2^k \) out of \( 2^n \) are used as code words.
  • \( r=n-k \) redundant bits

• **The Code Rate** of the coded word is \( R = k/n \).

• For every \( \overline{c}_i, \overline{c}_j \in C, i \neq j, d_H(\overline{c}_i, \overline{c}_j) \) is the **Hamming distance** between the two code words.

The **Hamming Distance** is defined as the number of bits which are different in the two code words.

• There is at least one pair of code words for which the distance is the least. This is called the **minimum Hamming distance of the code**.
Error Detection and Error Correction Capability (cont’d)

Example 1 (Repetition Code):

\[ G(0) \rightarrow 000 \]
\[ G(1) \rightarrow 111 \]

\( n = 3, \; k = 1, \; r = n - k = 2 \)
\[ d_H(000, 111) = w_H(111) = d_{\text{min}} = 3 \]

\( w_H \) is the Hamming Weight of a code word, defined as the number of “1” bits in the code word (the Hamming distance between the code word and the zero code word).
Error Detection and Error Correction Capability (cont’d)

Example 1 (cont’d):

Decision: based on the minimum Hamming distance between the received word and the code words.

- The code corrects 1 error \(d_H=1\), but does not simultaneously detect the 2 bit error. Moreover, we can miscorrect the received word.

- The code detects up to two bits in error (3 bits in error lead to a code word; \(d_{\text{min}}\) between the two code words is 3).

<table>
<thead>
<tr>
<th>Received Word</th>
<th>Decoded Word</th>
<th>Error Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>
Error Detection and Error Correction Capability (cont’d)

Example 2 (Repetition Code):

<table>
<thead>
<tr>
<th>Received Word</th>
<th>Decoded Word</th>
<th>Received Word</th>
<th>Decoded Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>1001</td>
<td>Error</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>1010</td>
<td>Error</td>
</tr>
<tr>
<td>0011</td>
<td>Error</td>
<td>1011</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>1100</td>
<td>Error</td>
</tr>
<tr>
<td>0101</td>
<td>Error</td>
<td>1101</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>Error</td>
<td>1110</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>1111</td>
<td>1</td>
</tr>
</tbody>
</table>

$G(0) \rightarrow 0000$
$G(1) \rightarrow 1111$

$n = 4$
$k = 1$
$r = 3$
$d_{\text{min}} = 4$

- Correct 1 error ($d_H = 1$) and Detect 2 errors ($d_H = 2$)
- An error of 3 or 4 bits will be miscorrected.
Error Detection and Error Correction Capability (cont’d)

Hamming Distance and Code Capability

- **Detect Up to** $t$ **Errors** IF AND ONLY IF $d_{\text{min}} \geq t + 1$

  **Example:** Repetition Code, $n = 3$, $k = 1$, $r = 2$, $d_{\text{min}} = 3$. This code detects up to $t = 2$ errors.

- **Correct Up to** $t$ **Errors** IF AND ONLY IF $d_{\text{min}} \geq 2t + 1$

  **Example:** Repetition Code, $n = 3$, $k = 1$, $r = 2$, $d_{\text{min}} = 3$. This code corrects $t = 1$ error.

- **Detect Up to** $t_d$ **Errors** and **Correct Up to** $t_c$ **Errors** IF AND ONLY IF

  $$d_{\text{min}} > 2t_c + 1 \quad \text{and} \quad d_{\text{min}} \geq t_c + t_d + 1$$

  **Example:** Repetition Code, $n = 3$, $k = 1$, $r = 2$, $d_{\text{min}} = 3$. This code cannot simultaneously correct ($t_c = 1$) and detect ($t_d = 2$) errors.
Error Detection and Error Correction Capability (cont’d)

The minimum Hamming Distance is related to the number of redundant bits, \( r \)

\[
 d_{\text{min}} \leq r + 1
\]

This gives us the lower limit on the number of the redundant bits for a certain minimum Hamming distance (certain detection and correction capability), and it is called the **Singleton Bound**.

\[
 r \geq d_{\text{min}} - 1
\]

**Example:**

Repetition Code, \( n = 3, \ k = 1, \ r = 2, \ d_{\text{min}} = 3 \).

\[
 d_{\text{min}} = r + 1
\]

See its error detection and correction capabilities as previously discussed.
Codes: Classification

Types of Codes (we discuss about)

- Linear Block Codes
- Convolutional Codes
Linear Block Codes can be mathematically treated using the mathematics of vector spaces.
ERROR DETECTION and CORRECTION

- **Linear Block Codes**

\[(A, +, \bullet)\]  
Galois Field with two elements  
\[A = \{0, 1\}, A = GF(2)\]

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Exclusive Or (Digital Logic)

<table>
<thead>
<tr>
<th>\cdot</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

And (Digital Logic)

\[(A^n, +, \bullet)\]  
Vector space, where \(A^n\) is a set with elements  
\[\bar{a} = (a_0, \ldots, a_{n-1}), \text{ with each } a_i \in A\]

- Vector Addition
- Scalar Multiplication
Linear Block Codes (cont’d)

The **set of code words,** \( C \), **is a subset of** \( A^n \).

It is a **subspace** (\( 2^k \) elements); any subspace is also a vector space.

\[
\text{If } \overline{c}_1, \overline{c}_2 \in C \Rightarrow \overline{c}_1 + \overline{c}_2 \in C
\]

The sum of two code words is also a code word (taken as the definition of a linear code).

**Consequence :**

- All-zero vector is a code word, \( \overline{0} \in C \) (because \( \overline{c}_1 + \overline{c}_1 = \overline{0} \))
Linear Block Codes (cont’d)

Linear Independent: \( \overline{c}_0, \ldots, \overline{c}_{k-1} \)

\[
a_0 \overline{c}_0 + \ldots + a_{k-1} \overline{c}_{k-1} = 0 \quad \text{if and only if} \quad a_0 = \ldots = a_{k-1} = 0
\]

Basis: \( \overline{c}_0, \ldots, \overline{c}_{k-1} \)

If they are linear independent and if and only if every \( \overline{c} \in C \) can be uniquely written as

\[
\overline{c} = a_0 \overline{c}_0 + \ldots + a_{k-1} \overline{c}_{k-1}
\]

The **Dimension** of a vector space is defined as the number of basis vectors is takes to describe (span) it.
Linear Block Codes (cont’d)

HOW DO WE GENERATE A CODE WORD?

\[ \bar{c} = \bar{m}G \]

Code Word \(1 \times n\) \hspace{1cm} Message \(1 \times k\) \hspace{1cm} Generator Matrix \(k \times n\)

Linear Combination of the rows of the \(G\) matrix → The \(k\)th rows must be linearly independent. They form a basis.

\[ \bar{c} = m_0 \bar{g}_0 + \ldots + m_{k-1} \bar{g}_{k-1} \]
Linear Block Codes (cont’d)

All the lines of \( G \) are code words!

Ex: \( \bar{m} = (1,0,...,0) \rightarrow \bar{c} = \bar{g}_0 \)

Example 3 : \( n = 7, k = 4, r = 3 \)

\[
(c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6) = (m_0 \ m_1 \ m_2 \ m_3)
\]

**Generator Matrix for the Linear Block Code (7,4)**

\[
G = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{bmatrix} = \begin{bmatrix}
\bar{g}_0 \\
\bar{g}_1 \\
\bar{g}_2 \\
\bar{g}_3 \\
\end{bmatrix}
\]

\[
\bar{c} = \bar{m}G = m_0 \bar{g}_0 + m_1 \bar{g}_1 + m_2 \bar{g}_2 + m_3 \bar{g}_3
\]
### ERROR DETECTION and CORRECTION

#### Linear Block Codes (cont’d)

**Linear Block Code with** $k = 4$ **and** $n = 7$.

<table>
<thead>
<tr>
<th>Decimal number</th>
<th>$m_0$</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$c_0$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Error Detection and Correction

- Linear Block Codes (cont’d)

Example 3 (cont’d):

- Generator Matrix for the Linear Systematic Block Code (7,4)

\[ n = 7, \ k = 4, \ n - k = 3 \]

\[
G = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
= [PL_{4x4}] 
\]
ERROR DETECTION and CORRECTION

Linear Block Codes (cont’d)

Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)

\[
(c_0, c_1, c_2, c_3, c_4, c_5, c_6) = (m_0, m_1, m_2, m_3)
\]

\[
\begin{align*}
\begin{cases}
    c_0 = m_0 + m_2 + m_3 \\
    c_1 = m_0 + m_1 + m_2 \\
    c_2 = m_1 + m_2 + m_3 \\
    c_3 = m_0 \\
    c_4 = m_1 \\
    c_5 = m_2 \\
    c_6 = m_3
\end{cases}
\end{align*}
\]

\[
\begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

\( \Rightarrow \) ENCODING CIRCUIT
A linear block code with such a structure is called **Linear Systematic Block Code**.
ERROR DETECTION and CORRECTION

❖ Linear Block Codes (cont’d)

Input $m_0 \ m_1 \ m_2 \ m_3$

Message Register

$m_0$ $m_1$ $m_2$ $m_3$

Parity Register

$c_0 \ c_1 \ c_2 \ m_0 \ m_1 \ m_2 \ m_3$

ENCODING CIRCUIT FOR THE (7,4) LINEAR SYSTEMATIC BLOCK CODE

6876: Communication Networks
Linear Block Codes (cont’d)

Hamming Weight and Hamming Distance

\[ d_H(\overline{c}_1, \overline{c}_2) = w_H(\overline{c}_3) \]

Example 3: \( n = 7, k = 4, r = 3 \)

\[ \overline{c}_1 = (1101000) \]
\[ \overline{c}_2 = (0110100) \]
\[ d_H(\overline{c}_1, \overline{c}_2) = 4 \]
\[ \overline{c}_1 + \overline{c}_2 = (1011100) = \overline{c}_3 \]
\[ \rightarrow d_H(\overline{c}_1, \overline{c}_2) = w_H(\overline{c}_3) \]
\[ w_H(\overline{c}_3) = 4 \]
Linear Block Codes (cont’d)

Hamming Weight and Hamming Distance (cont’d)

\[
\min_{\begin{array}{c} i, j=1,\ldots,2^k \\
i \neq j \end{array}} d_H(\overline{c}_i, \overline{c}_j) = \min_{i=1,\ldots,2^k} w_H(\overline{c}_i)
\]

The minimum Hamming distance of a linear block code is equal to the minimum Hamming weight of the non-zero code vectors.

Example 3: \( n = 7, k = 4, r = 3, d_{\text{min}} = w_{\text{min}} = 3 \)
Linear Block Codes (cont’d)

Received Vectors

\[ \overline{v} = \overline{c} + \overline{e} \]

Vector Error

No Error

\[ \overline{e} = (00000000) \]

Ex: An error at the first bit

\[ \overline{e} = (10000000) \]
Linear Block Codes (cont’d)

Parity Check Matrix

\[ \mathbf{G} \mathbf{H}^T = \mathbf{0} \]

\( \mathbf{G} = \) Generator Matrix \( k \times n \)

\( \mathbf{H} = \) Parity Check Matrix \( n-k \times n \)

Systematic Code:

\[ \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T] \]

For a Code Word:

\[ \overline{c} \mathbf{H}^T = \overline{m} \mathbf{G} \mathbf{H}^T = \mathbf{0} \]
Linear Block Codes (cont’d)

Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)

- **Parity-Check Matrix for the Linear Systematic Block Code (7,4)**

\[
H = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix} = \begin{bmatrix} I_{3 \times 3} \end{bmatrix}^T
\]
Linear Block Codes (cont’d)

Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)

\[
G H^T = 0
\]

\[
\bar{c} H^T = 0
\]

\[
(c_0 c_1 c_2 m_0 m_1 m_2 m_3) H^T = 0
\]

\[
\begin{cases}
c_0 + m_0 + m_2 + m_3 = 0 \\
c_1 + m_0 + m_1 + m_2 = 0 \\
c_2 + m_1 + m_2 + m_3 = 0
\end{cases}
\]

Parity Check Equations
Error Detection and Correction

Linear Block Codes (cont’d)

Example 3: \( n = 7, \ k = 4, \ r = 3 \)

\[
\begin{align*}
\bar{c} & = (0010111) \quad \text{(Code word #14 in table)} \\
\bar{e} & = (0001101) \\
\bar{v} & = (0011010) \quad \text{(Code word #5 in table, !!!)}
\end{align*}
\]

There is an error, but this error is undetectable!

The error vector introduces 3 errors. But the minimum Hamming distance for this code is 3, and, such, a 3 error pattern can lead to another code word!
Linear Block Codes (cont’d)

Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)

What is the capability of this code to correct and detect errors?

- **Detect Up to** \( t \) **Errors** IF AND ONLY IF \( d_{\text{min}} \geq t + 1 \)
- **Correct Up to** \( t \) **Errors** IF AND ONLY IF \( d_{\text{min}} \geq 2t + 1 \)
- **Detect Up to** \( t_d \) **Errors** and **Correct Up to** \( t_c \) **Errors** IF AND ONLY IF \( d_{\text{min}} > 2t_c + 1 \) and \( d_{\text{min}} \geq t_c + t_d + 1 \)

The minimum Hamming distance is 3, and, such, the number of errors which can be detected is 2 and the number of errors which can corrected is equal to 1.

The code does not have the capability to simultaneously detect and correct errors. (see the relations between \( d_{\text{min}} \) and the correction/detection capability of a code).
Linear Block Codes (cont’d)

Example 3 (cont’d): $n = 7$, $k = 4$, $r = 3$

When we say that the number of errors which can be detected is 2, we refer at all error patterns with 2 bits in errors. However, the code is capable to detect patterns with more than 2 errors, but not all!

What is the number of error patterns which can be detected with this code?

The total number of error patterns is $2^n - 1$ (the all-zero vector is not an error!). However, $2^k - 1$ of them lead to code words, which mean that they are not detectable.

So, the number of error patterns which are detectable is $2^n - 2^k$. 
† Linear Block Codes (cont’d)

Syndrome Decoder

✓ Syndrome: \[ \overline{\mathbf{s}} = \overline{\mathbf{v}} \mathbf{H}^\mathsf{T} \]

\[ = \overline{\mathbf{0}} \quad \text{if} \quad \overline{\mathbf{v}} = \overline{\mathbf{c}} \]

\[ \neq \overline{\mathbf{0}} \quad \text{if} \quad \overline{\mathbf{v}} \neq \overline{\mathbf{c}} \]

USED FOR DETECTION!

\[ \overline{\mathbf{s}} = (\overline{\mathbf{c}} + \overline{\mathbf{e}}) \mathbf{H}^\mathsf{T} = \overline{\mathbf{e}} \mathbf{H}^\mathsf{T} \]

The syndrome is independent on the code word;
It depends only on the error vector.
* Linear Block Codes (cont’d)

**Syndrome Decoder and Standard Array**

- The first word in the first column of the array is the zero code-word (it also means zero error).
- For no error, the code words are received. These are given in the first row of the array. So, the first row contains $2^k$ code words, including the zero code-word.
- All $2^n$ words are contained in the array. Each row contains $2^k$ words. So, the number of columns is $2^k$. The number of rows will then be $2^n/2^k=2^{n-k}$.
- Each row is called coset. In the first column we have all correctable error patterns. These are called coset leaders.
- Decoding is correctly done if and only if the error pattern caused by the channel is a coset leader (including the zero-vector).
Linear Block Codes (cont’d)

Syndrome Decoder and Standard Array (cont’d)

- To minimize the probability of a decoding error, the error patterns that are more likely to occur for a given channel should be chosen as coset leaders.

- For a binary symmetric channel, an error pattern of smaller weight is more probable than an error pattern of larger weight. Therefore, when the standard array is formed, each coset leader should be chosen to be a vector of at least weight from the remaining available vectors. Choosing coset leaders this way, each coset leader will have the minimum weight in its coset.

- The words on each column, except for the first element, which is a code word, are obtained by adding the coset leader to the code word. This way, in a column, one gets the words which are at minimum distance of the code word, which is the first element of the column.

- A linear block code is capable to correct $2^{n-k}$ error patterns.
## STANDARD ARRAY

<table>
<thead>
<tr>
<th>Coset Leader</th>
<th>$\overline{c}_0 = \overline{0}$</th>
<th>$\overline{c}_1$</th>
<th>...</th>
<th>$\overline{c}_i$</th>
<th>...</th>
<th>$\overline{c}_{2^k-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{e}_1$</td>
<td>$\overline{e}_1 + \overline{c}_1$</td>
<td>...</td>
<td>$\overline{e}_1 + \overline{c}_i$</td>
<td>...</td>
<td>$\overline{e}<em>1 + \overline{c}</em>{2^k-1}$</td>
<td></td>
</tr>
<tr>
<td>$\overline{e}_2$</td>
<td>$\overline{e}_2 + \overline{c}_1$</td>
<td>...</td>
<td>$\overline{e}_2 + \overline{c}_i$</td>
<td>...</td>
<td>$\overline{e}<em>2 + \overline{c}</em>{2^k-1}$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>$\overline{e}_{2^{n-k}-1} + \overline{c}_1$</td>
<td>...</td>
<td>$\overline{e}_{2^{n-k}-1} + \overline{c}_i$</td>
<td>...</td>
<td>$\overline{e}<em>{2^{n-k}-1} + \overline{c}</em>{2^k-1}$</td>
<td></td>
</tr>
</tbody>
</table>

## STANDARD ARRAY FOR THE (7,4) LINEAR SYSTEMATIC BLOCK

<table>
<thead>
<tr>
<th>0000000</th>
<th>1101000</th>
<th>0110100</th>
<th>1011000</th>
<th>1110010</th>
<th>0011010</th>
<th>1001100</th>
<th>0101110</th>
<th>1010010</th>
<th>0111010</th>
<th>1100110</th>
<th>0010111</th>
<th>1001101</th>
<th>0101111</th>
<th>1111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>0101000</td>
<td>1110100</td>
<td>0011010</td>
<td>1011010</td>
<td>0111010</td>
<td>1100111</td>
<td>0010001</td>
<td>1111101</td>
<td>0100110</td>
<td>1001101</td>
<td>0110110</td>
<td>1101111</td>
<td>0011111</td>
<td>1111111</td>
</tr>
<tr>
<td>0100000</td>
<td>1001000</td>
<td>0010100</td>
<td>1111010</td>
<td>0010110</td>
<td>1100110</td>
<td>0101110</td>
<td>1000110</td>
<td>0111010</td>
<td>1100110</td>
<td>0010011</td>
<td>1110101</td>
<td>0110110</td>
<td>1100111</td>
<td>0011110</td>
</tr>
<tr>
<td>0010000</td>
<td>1111000</td>
<td>0100100</td>
<td>1010110</td>
<td>0010111</td>
<td>1010110</td>
<td>0111110</td>
<td>1000111</td>
<td>0110010</td>
<td>1111011</td>
<td>0110011</td>
<td>1101110</td>
<td>0011110</td>
<td>1111110</td>
<td></td>
</tr>
<tr>
<td>0001000</td>
<td>1100000</td>
<td>0110100</td>
<td>1010010</td>
<td>0010111</td>
<td>1001110</td>
<td>0110010</td>
<td>1100111</td>
<td>0110011</td>
<td>1101110</td>
<td>0011110</td>
<td>1111111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000100</td>
<td>1110100</td>
<td>0011000</td>
<td>1011000</td>
<td>0111010</td>
<td>1100010</td>
<td>0010110</td>
<td>1010111</td>
<td>0111011</td>
<td>1100110</td>
<td>0010011</td>
<td>1101110</td>
<td>0111101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000010</td>
<td>1110100</td>
<td>0111010</td>
<td>0111101</td>
<td>1100010</td>
<td>0010100</td>
<td>1010110</td>
<td>0111011</td>
<td>1100110</td>
<td>0010011</td>
<td>1101110</td>
<td>0111101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000001</td>
<td>1101001</td>
<td>0110101</td>
<td>1011101</td>
<td>1110011</td>
<td>0011011</td>
<td>1000111</td>
<td>0101111</td>
<td>1010001</td>
<td>0111011</td>
<td>1100110</td>
<td>0010010</td>
<td>1101110</td>
<td>0111101</td>
<td></td>
</tr>
</tbody>
</table>
Linear Block Codes (cont’d)

**The Minimum (Hamming) Distance and Error Correction Capability of a Linear Block Code**

\[ t = \left\lfloor \frac{d_{\text{min}} - 1}{2} \right\rfloor \]

For \((7,4): d_{\text{min}}=3\) and \(t=1\).
Linear Block Codes (cont’d)

Syndrome Decoder and Standard Array (cont’d)

✓ All the $2^k n$-tuples ($n$ bit words) of a coset have the same syndrome.

Steps in the Syndrome Decoder

1. For the received word, the syndrome is calculated.
   $$\overline{s} = \overline{v}H^T = \overline{e}H^T$$

2. The coset leader $\overline{e}$ is calculated.

3. The transmitted code word is obtained.
   $$\overline{c} = \overline{v} + \overline{e}$$
Linear Block Codes (cont’d)

Syndrome Decoder and Standard Array (cont’d)

![Diagram of Syndrome Decoder and Standard Array](image)

General decoder for a linear block code.
Linear Block Codes (cont’d)

Syndrome Decoder and Standard Array (cont’d)
Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)

**Syndrome Look-Up Table (Decoding Table) for the (7,4) Linear Block Code**

<table>
<thead>
<tr>
<th>Syndrome</th>
<th>Coset Leader (Correctable Error Pattern)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 s_1 s_2 )</td>
<td>( e_0 \ e_1 \ e_2 \ e_3 \ e_4 \ e_5 \ e_6 )</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>
Linear Block Codes (cont’d)

Syndrome Decoder and Standard Array (cont’d)

Example 3 (cont’d): \( n = 7, \ k = 4, \ r = 3 \)
ERROR DETECTION and CORRECTION

- Linear Block Codes (cont’d)

**EXAMPLES:**

<table>
<thead>
<tr>
<th>$(n, k)$</th>
<th>$t$</th>
<th>$R = k/n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,1)</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>(7,4)</td>
<td>1</td>
<td>0.57</td>
</tr>
<tr>
<td>(31,26)</td>
<td>1</td>
<td>0.838</td>
</tr>
<tr>
<td>(10,4)</td>
<td>2</td>
<td>0.4</td>
</tr>
<tr>
<td>(15,8)</td>
<td>2</td>
<td>0.533</td>
</tr>
<tr>
<td>(10,2)</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>(23,12)</td>
<td>3</td>
<td>0.52</td>
</tr>
</tbody>
</table>

Here $t$ is the number of errors which can be corrected.
Cyclic Codes

Is a class of linear block codes, which can be implemented with extremely cost effective electronic circuits.

Cyclic Shift Property

A cyclic shift of $\overline{c} = (c_0 c_1 \ldots c_{n-2} c_{n-1})$ is given by $\overline{c}^{(1)} = (c_{n-1} c_0 c_1 \ldots c_{n-2})$

In general, a cyclic shift of $\overline{c}$ can be written as $\overline{c}^{(i)} = (c_{n-i} c_{n-i+1} \ldots c_{n-1} c_0 c_1 \ldots c_{n-i-1})$
Cyclic Codes (cont’d)

Definition:

A cyclic code is a linear block code $C$, with code words $\overline{c} = (c_0c_1\ldots c_{n-2}c_{n-1})$, such that for every $\overline{c} \in C$, the vector given by the cyclic shift of $\overline{c}$ is also a code word.

Example 1:

The (6,2) repetition code $C = \{(000000), (111111), (010101), (101010)\}$ is a cyclic code, since a cyclic shift of any of its code vectors results in a vector that is element of $C$. 
Cyclic Codes (cont’d)

Example 2:

The (5,2) linear block code defined by the generator matrix

\[ G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \]

is not a cyclic code.

Its code vectors are

00000
10111
01101
11010

(rows of G are code words and so their sum)

The cyclic shift of (10111) is (11011), which is not an element of \( C \).
Similarly, the cyclic shift of (01101) is (10110), which is also not a code word.
Cyclic Redundancy Check (CRC) Codes

- CRC are error-detecting codes typically used in ARQ systems.
- CRC have no error correction capability, but they can be used in combination with an error-correcting code.
- The error control system is in the form of a concatenated code.
Convolutional Codes

- Are the second major form of error-correcting channel codes.
- Differ from the linear block codes in both structural form and error correcting properties.

With linear block codes, \( k \) binary digits are transmitted as an \( n \)-bit code word. On the other hand, convolutional codes convert the entire data stream into a single code word.

The code rate for the linear block codes can be \( \geq 0.95 \), but they have limited error correction capabilities. For convolutional codes, the code rate is usually below 0.9, but they have more powerful error-correcting capabilities. Puncturing is used to achieve higher code rates.
Convolutional Codes (cont’d)

**Basic Idea:** the source data is broken into frames of $k_0$ bits per frame. $M+1$ frames of source data are coded into $n_0$-bit code frame, where $M$ is the **memory depth** of the shift register.

Convolutional codes are encoded using shift registers. As each new data frame is read, the old data is shifted one frame to the right, and a new code word is calculated.

**Characteristics of the Code:**

- **Code Rate**
  $$ R = \frac{k_0}{n_0} $$

- **Constraint Length**
  $$ \nu = M + 1 $$

For binary convolutional codes: $k_0=1$
Convolutional Codes (cont’d)

Basic Idea (cont’d)

Figure: Shift Register Encoder
Convolutional Codes (cont’d)

Example: \( R = \frac{1}{2} \quad \nu = 3 \)

For each bit on the input, we obtain 2 bits on the output. These are interleaved and sent as a two bit symbol sequence.

We can associate two code polynomials, i.e.,

\[
\begin{align*}
g_0(X) &= 1 + X + X^2 \\
g_1(X) &= 1 + X^2
\end{align*}
\]

such that

\[
\begin{align*}
c_0(X) &= m(X)g_0(X) \\
c_1(X) &= m(X)g_1(X)
\end{align*}
\]
Convolutional Codes (cont’d)

Convolutional codes provide very powerful error correction capability, at the price of low code rate.

The decoding process is more complex when compared with linear block codes (Viterbi algorithm).

Performance of convolutional codes are determined by the minimum free distance (a Hamming distance, \(d_f\)).
Convolutional Codes (cont’d)

Examples:

**Codes with rate 1/2.** The following codes were discovered by Odenwalder (1970) and Larsen (1972):

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generator Polynomials</th>
<th>$d_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5,7)</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>(15,17)</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>(23,35)</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>(53,75)</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>(133,171)</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>(247,371)</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>(561,753)</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>(1167,1545)</td>
<td>12</td>
</tr>
</tbody>
</table>

**Codes with rate 1/3.** The following codes were discovered by Odenwalder (1970) and Larsen (1972):

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generator Polynomials</th>
<th>$d_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5,7,7)</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>(13,15,17)</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>(25,33,37)</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>(47,53,75)</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>(133,145,175)</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>(225,331,367)</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>(557,663,711)</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>(1117,1365,1633)</td>
<td>20</td>
</tr>
</tbody>
</table>
Convolutional Codes (cont’d)

Examples:

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generator Polynomials</th>
<th>$d_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5, 7, 7, 7)</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>(13, 15, 15, 17)</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>(25, 27, 33, 37)</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>(53, 67, 71, 75)</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>(133, 135, 147, 163)</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>(235, 275, 313, 357)</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>(463, 535, 733, 745)</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>(1117, 1365, 1633, 1653)</td>
<td>27</td>
</tr>
</tbody>
</table>
Punctured Convolutional Codes

Idea: We start with a $1/n_0$ convolutional code, such as a 1/2 code rate. The transmitted code word corresponding to the 1/2 code is $(c_0c_1)$. We delete one of the code bits every 2 code symbols.

Code sequence: $(c_0c_1)_t, (c_0-)_{t+1}, (c_0c_1)_{t+2}, (c_0-)_{t+3}, \ldots$

The deleted code bits are not transmitted. In average, 3 code bits are transmitted every two message bits, which yields a rate 2/3 code.

Deleting code bits is called puncturing the code.

The rate is increased at the expense of reducing the minimum free distance.
GOOD PUNCTURED CONVOLUTIONAL CODES

Examples:

\[(7,5),7 \implies g_0(X) = 1 + X + X^2 \quad \text{(7 in octal)} \implies M = 2\]

\[g_1(X) = 1 + X^2 \quad \text{(5 in octal)}\]

The second message bit is encoded using only the generator polynomial 7.

\[\Rightarrow R = \frac{2}{3}\]

Code sequence: \[(c_0c_1)_t, (c_0c_1)_{t+1}, (c_0c_1)_{t+2}, (c_0c_1)_{t+3}, \ldots\]

The Puncturing Period: the number of bits encoded before the returning to the base code.

Here the puncturing period is equal to 2.
● Good Punctured Convolutional Codes

Examples:

\[(15,17),15,17 \implies g_0(X) = 1 + X + X^3 \quad \text{(15 in octal)} \implies M = 3\]

\[g_1(X) = 1 + X + X^2 + X^3 \quad \text{(17 in octal)} \implies R = \frac{3}{4}\]

The second message bit is encoded using only the generator polynomial 15, whereas the third message bit only by using the generator polynomial 17.

Code sequence: \((c_0c_1)_t, (c_0-)_t+1, (-c_1)_t+2, (c_0c_1)_t+3, \ldots\)

Here the **puncturing period** is equal to 3.
**Good Punctured Convolutional Codes**

The punctured codes presented here are punctured versions of known good-rate 1/2 codes.

However, it is not always true that puncturing a good code \((1/n_0 \text{ rate})\) yields a good punctured code.

There is no known systematic procedure for generating good punctured convolutional codes. Good codes are discovered by computer search. Examples are given in the sequel.

**Examples:**

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generators (Octal Representation)</th>
<th>(d_f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(7, 5), 7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>(15, 13), 15</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>(31, 33), 31</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>(73, 41), 73</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>(163, 135), 163</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>(337, 251), 337</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>(661, 473), 661</td>
<td>8</td>
</tr>
</tbody>
</table>
## ERROR DETECTION and CORRECTION

### Good Punctured Convolutional Codes

**Rate 3/4 Punctured Codes:** (Discovered by Cain et al.)

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generators (Octal Representation)</th>
<th>$d_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(5, 7), 5, 7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>(15, 17), 15, 17</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>(35, 37), 37, 37</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>(61, 53), 53, 53</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>(135, 163), 163, 163</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>(205, 307), 307, 307</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>(515, 737), 737, 737</td>
<td>6</td>
</tr>
</tbody>
</table>

**Rate 4/5 Punctured Codes:** (Published by Lee, 1988)

<table>
<thead>
<tr>
<th>Constraint Length</th>
<th>Generators (Octal Representation)</th>
<th>$d_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(17, 11), 11, 11, 13</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>(37, 35), 25, 37, 23</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>(61, 53), 47, 47, 53</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>(151, 123), 153, 151, 123</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>(337, 251), 237, 237, 235</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>(765, 463), 765, 765, 473</td>
<td>5</td>
</tr>
</tbody>
</table>