During transmission in non-ideal channels, some bits are changed and received with errors. It is a very important task to handle those errors properly in the receiver. Therefore, error detection and/or error correction are used in all practical digital transmission systems to increase the reliability. These methods help in exploiting the channel capacity in an efficient way with increased reliability.

The basics of error control coding are presented in this part of lecture notes. This is very important and very wide area. In the next slides, only the basic principles of some important coding methods are considered and their importance to digital the transmission system is highlighted.

This part covers the following topics:
- error handling methods
- hard/soft decoding
- block codes
- convolution codes
- trellis codes

Source: Lee & Messerschmitt, Chapter 11.
The error handling task is done by channel coding, before line coding.

Two different approaches exist:

- **Error detection**: The code can be designed such it detects, for example, all single bit-errors.

  After an error is detected using error detection code, the receiver requests retransmission of the part of signal which is received with errors.

  Error detection can be used in combination with service monitoring.

- **Error correction**: This class of codes detect and correct detected transmission error using redundancy.

All coding methods are based on adding some redundancy into the signal.

  - The first way to add redundancy is to insert additional symbols to the transmitted symbol stream, which will also increase the symbol rate.

  - The second approach is to increase the number of symbols in the alphabet while keeping the symbol rate the same.
Hard/Soft Decoding

**Hard decoding**

In this case, the slicer makes a "hard decision", in order to detect the transmitted symbols (complete with their redundancy).

From the perspective of the decoder, the channel is binary and makes transmission errors, a binary symmetric channel (BSC) model can be used.

**Soft decoder**

In this case, the decoder makes decisions on the information bits without making intermediate decisions about transmitted symbols. Soft decoding eliminates the use of a slicer.

Soft decoding gives better performance at the expense of implementation complexity, because it makes use of the information that the slicer throws away.
Classes of Codes

Block codes

Blocks of $k$ source bits are mapped into blocks of $n$ coded bits, thus $n-k$ bits are added.

The block code has a code rate $\frac{k}{n}<1$.

Block codes are usually used with hard decoding, but soft decoding may also be possible.

Convolutional codes

Redundancy is added without dividing the source bits into blocks, in a continuous manner; the coded bits are produced at higher rate.

They can be used with either hard decoding or soft decoding.

Trellis codes (signal space codes)

Redundancy is added, for example, by using 64-QAM constellation for 16 symbol source alphabet.

It is used only with soft decoding.
Purpose of Coding (from Benedetto & Biglieri)

Here A, B and C correspond to uncoded, soft-decoding, and hard-decoding cases, respectively.
Coding Gain

In the Lee-Messerschmitt book, coding gain is defined as the reduction (in dB’s) of the required SNR to achieve a given bit error rate (BER).

Block codes and convolutional codes with error correction capabilities add a considerable amount of redundancy. They increase the symbol rate and required bandwidth, if the line-coding and modulation method is unchanged. Consequently, in the case of AWGN channel, the noise bandwidth is also increased, and thus the SNR becomes worse if the signal power is the same. This has an effect on the useful coding gain.

For trellis codes, the noise bandwidth does not change, and here the coding gain tells directly how much the transmitted signal power can be decreased due to coding.
About the Concept of Coding Gain

In the Lee-Messerschmitt book, coding gain is defined based on the SNR, and when using this definition, one has always to pay attention to the effect of increased (noise) bandwidth, resulting in reduced SNR, when evaluating the benefit of coding.

In other literature, coding gain is quite often defined using the $E_b/N_0$ ratio, and the resulting coding gain tells the true benefit of the code. In this context, it is important to note that $E_b$ is the energy in the signal corresponding to one source bit. For a rate $R$ code, one source bit energy is $1/R$ times the channel bit energy.

With these definitions, the BER in uncoded (u) and coded (c) cases are the same iff (see examples later)

\[
d_{\text{min}}^2 \frac{SNR_c}{SNR_u} = d_{\text{min}}^2 \frac{E_{bc} R_c}{N_0 W_c} = \frac{E_{bu} r_u}{N_0 W_u} = SNR_u
\]

which is equivalent to

\[
R d_{\text{min}}^2 \frac{E_b c}{N_0} = E_b u / N_0
\]

because for fixed constellation and modulation parameters, the ratio of bit-rate and bandwidth is constant.

Thus we obtain:

\[
\text{Coding gain} = 10 \log_{10}(R d_{\text{min}}^2)
\]

which takes automatically into account the effects of different noise bandwidths.
Block Codes

(n, k)-code representation: k source bits n coded bits.

A parity check code is fully characterized by its generator matrix, G:

\[ c = bG \]

where \( b \) is row vector of source bits
\( c \) is row vector of coded bits (codeword)
\( G \) is generator matrix, consisting of zeros and ones.

A code is set of all possible codewords. Every codeword is a modulo two summation of the rows of the generator matrix.

=> The zero vector is a codeword.
    The modulo-2 sum of any two codewords is also codeword.

The code is systematic if the \( k \) first bits of the codeword are exactly the source bits. The generator matrix of a systematic code has the following form:

\[ G = [I_k \mid P] \]

Here \( I_k \) is \( k \times k \) identity matrix.

All parity check codes are linear and all linear block codes are parity check codes with some generator matrix. All linear block codes are also equivalent to a systematic code, obtained by reordering the coded bits.
A simple parity check code \((n, n-1)\) (or \((k+1, k)\))

\[
C^{(1)} = B^{(1)}, \quad C^{(2)} = B^{(2)}, \ldots, \quad C^{(k)} = B^{(k)}
\]

\[
C^{(n)} = C^{(k+1)} = B^{(1)} \oplus B^{(2)} \oplus \ldots \oplus B^{(k)}
\]

Symbol \(\oplus\) means modulo-2 sum (exclusive-or).

An extra bit is appended to ensure that there is an even number of ones in the codeword.

The generator matrix is:

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

This code is systematic.

(For later reference:)

The minimum Hamming weight = minimum Hamming-distance = 2

The minimum Euclidean distance is \(d_E = 2a\sqrt{2}\).
(7,4) Hamming-code:

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

The code is systematic.

(For later reference:)

Minimum Hamming-distance = 3

Minimum Euclidean distance is \( d_E = 2a\sqrt{3} \)
Hamming and Euclidean Distance

The decoding principle can be based on the following:

Choose the codeword that is closest to the received one based on:

- Euclidean distance for soft decoding
- Hamming distance hard decoding.

The probability of error is dominated by the two codewords closest in Euclidean or Hamming distance measure.

In the following, the important thing to remember is fact the soft decoding is superior. This comparison assumes that binary antipodal signalling is used.

In this case the relation between Euclidean and Hamming distances can be expressed as:

\[ d_E = 2a \sqrt{d_H} \]

In the case of linear code

\[ d_{H,\text{min}} = \min_{\substack{w_H(c) \text{ codeword's Hamming-weight, i.e., the number of 1-bits.}}} \]

\[ \min_{c \in \mathcal{C}} w_H(c) \quad \text{where } w_H(c) \]
Performance of Soft Decoders

The input to the soft decoder is the sample stream before applying it to any slicer.

Assume the AWGN case and zero ISI, and binary antipodal symbols.

The observation vector is

\[ q = a + n \]

where \( a \) is a vector of antipodal symbols that corresponds to the binary codeword \( c \). \( n \) is vector noise samples. And the noise is white Gaussian with zero mean and variance \( \sigma_c^2 \).

As discussed earlier, an ML detector selects the vector closest to the observation \( q \) in Euclidean distance, and the following is satisfied:

\[
(2^k - 1)Q\left[ \frac{d_{E,\text{min}}}{2\sigma_c} \right] \geq \Pr[\text{block error}] \geq Q\left[ \frac{d_{E,\text{min}}}{2\sigma_c} \right]
\]

where \( d_{E,\text{min}} \) is minimum Euclidean distance of the code.

This equation shows the bounds for the probability of error in soft ML-based detection. The coarse upper bound is obtained by using union bound and assuming that all the codewords are at the minimum distance from each other. The lower bound is obtained by assuming that there is only one codeword at the the minimum distance from every codeword.
Each block error will produce at least one bit error so we can write:

\[
\Pr[\text{block error, soft decoding}] \geq \Pr[\text{bit error, soft decoding}]
\]

\[
\geq \frac{1}{k} \Pr[\text{block error, soft decoding}]
\]

resulting in

\[
(2^k - 1)Q\left[\frac{d_{E,\min}}{2\sigma_c}\right] \geq \Pr[\text{bit error, soft decoding}] \geq \frac{1}{k} Q\left[\frac{d_{E,\min}}{2\sigma_c}\right]
\]

The uncoded system has the following bit error probability:

\[
\Pr[\text{bit error, uncoded system}] = Q\left[\frac{a}{\sigma_u}\right]
\]

where \(\sigma_u^2\) is the noise variance at the slicer input for the uncoded system.

In general \(\sigma_u^2 \neq \sigma_c^2\).
Example I: Coding Gain

We consider here the simple parity check code \((n, n-1)\).

We make an approximation and use the multiplier 1 for the Q function when calculating the probability of error. We require the same probability of error in coded and uncoded systems:

\[
Q\left(\frac{d_{E,\text{min}}}{2\sigma_c}\right) = Q\left(\frac{2a_c\sqrt{2}}{2\sigma_c}\right) = Q\left(\frac{a_u}{\sigma_u}\right)
\]

where antipodal signal with amplitudes \(a_u\) and \(a_c\) in uncoded and coded cases, respectively. To get the same error rates, the amplitudes should be chosen to satisfy:

\[
a_c\sqrt{2}/\sigma_c = a_u/\sigma_u
\]

or \(SNR_u = 2SNR_c\). So there is 3 dB difference in the required SNRs.

This is not the whole story. We have to take into account the larger bandwidth due to increased symbol rate by \(n/(n-1)\) times. Because of this, the noise is increased. Requiring the same performance again:

\[
\sigma_c^2 = \frac{n}{n-1}\sigma_u^2
\]

\[
a_c^2/a_u^2 = 0.5 \cdot \frac{n}{n-1}
\]

In the case when \(n = 3\), the required signal level is 1.25 dB smaller compared to the uncoded case. The coding gain in this case is 1.25 dB, not 3 dB, because of increased noise bandwidth. (The same result can be obtained using the ideas mentioned on page 340.)
Example I: Coding Gain (cont.)

The lower and upper bounds for bit error probability are:

\[
\frac{1}{n-1} Q\left(\frac{a\sqrt{2(n-1)/n}}{\sigma_u}\right) \leq \Pr[\text{bit error, soft decoding}]
\]

\[
\leq (2^{n-1} - 1) Q\left(\frac{a\sqrt{2(n-1)/n}}{\sigma_u}\right)
\]

For a given probability of error, it is possible to find the power advantage numerically corresponding to the lower bound, upper bound, and crude approximation where the coefficient of the Q-function is ignored:

<table>
<thead>
<tr>
<th>Probability-of error</th>
<th>Best case</th>
<th>worst case</th>
<th>ignoring constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-5})</td>
<td>1.56 dB</td>
<td>0.77 dB</td>
<td>1.25 dB</td>
</tr>
<tr>
<td>(10^{-7})</td>
<td>1.45 dB</td>
<td>0.91 dB</td>
<td>1.25 dB</td>
</tr>
</tbody>
</table>

The bounds get closer as SNR increases.
Example II: Coding Gain

In this case, the minimum Hamming distance is 3, that is

\[ d_{E,\text{min}} = 2\sqrt{3} \]

\[ \frac{1}{4} \left[ \frac{a\sqrt{3}}{\sigma_c} \right] \leq \Pr[\text{bit error, soft decoding}] \leq 15 \left[ \frac{a\sqrt{3}}{\sigma_c} \right] \]

The reduction in the required SNR is \( 10 \log_{10} 3 = 4.7 \) dB.

If the increase when the noise bandwidth is taken into account:

\[ \sigma_c^2 = \frac{7}{4} \sigma_u^2 \]

\[ \frac{1}{4} \left[ \frac{a\sqrt{12/7}}{\sigma_u} \right] \leq \Pr[\text{bit error, soft decoding}] \leq 15 \left[ \frac{a\sqrt{12/7}}{\sigma_u} \right] \]

Ignoring the constant in front of the Q-function, the coding gain becomes:

\[ 10 \log_{10} \frac{12}{7} = 2.34 \text{ dB} \]

and the bounds are:

<table>
<thead>
<tr>
<th>Probability of error</th>
<th>best case</th>
<th>worst case</th>
<th>ignoring constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^{-5} )</td>
<td>3.00 dB</td>
<td>1.25 dB</td>
<td>2.34 dB</td>
</tr>
<tr>
<td>( 10^{-7} )</td>
<td>2.79 dB</td>
<td>1.56 dB</td>
<td>2.34 dB</td>
</tr>
</tbody>
</table>
In a hard decoder, the decoding is done after the slicer. The equivalent binary channel model (from the decoder point of view) is usually a BSC with probability of error $p$.

A hard-decoding ML-detector selects the codeword closest in Hamming distance to the observation.

All received bit vectors having at most

$$t = \left\lfloor \frac{(d_{H,\text{min}} - 1)}{2} \right\rfloor$$

(\lfloor \rfloor\ stands\ for\ rounding\ towards\ 0)

errors can be correctly decoded.

The code can correct $t$ errors per block.

A perfect code has the following properties:

- all bit patterns of length $n$ are within Hamming distance $t$ of a codeword, and
- no bit pattern of length $n$ is at Hamming distance $t$ or less from more than one code word.

Practically, this means that a perfect code can detect and correct $t$ error bits per block.

**Example 1:** $d_{H,\text{min}} = 2 \implies t = 0$

The code cannot correct any error.
Example 2: \[ d_{H,\text{min}} = 3 \Rightarrow t = 1 \]

The code can correct any single bit error in a codeword.

(7,4) Hamming code is perfect:

- All seven-bit patterns are either codewords or at one bit distance from exactly one codeword.
- Consequently, if a codeword is transmitted and two bit errors occur, then the received codeword will be at the distance of one from another codeword. Therefore, decoding error occurs.

For example:

- Sent codeword: \( \mathbf{c} = 0000000 \)
- Received bit vector: \( \mathbf{c} = 0001010 \)
- Decoded codeword: \( \hat{\mathbf{c}} = 0001011 \)
Perfect codes are optimal for BSC in the sense that they minimize the probability of error among all codes with the same $n$ and $k$.

With independent noise components, as in BSC model, the probability of $m$ errors in a block of $n$ bits is calculated using the binomial distribution:

$$ P(m,n) = \binom{n}{m} p^m (1-p)^{n-m} $$

A block error occurs if more than $t$ bit are in error:

$$ \Pr[\text{block error}] = \sum_{m=t+1}^{n} P(m,n) = 1 - \sum_{m=0}^{t} P(m,n) $$

This relation is exact for perfect codes, for non-perfect codes we can estimate an upper bound based on this relation (in this case some error patterns with more that $t$ bits can corrected using ML decoding).

Some practical codes are quasiperfect, meaning that they could correct $t+1$ errors (sometimes) but not $t+2$ errors. The following lower bound is obtained for this case:

$$ \sum_{m=t+2}^{n} P(m,n) \leq \Pr[\text{block error}] \leq \sum_{m=t+1}^{n} P(m,n) $$

In the literature, many different bounds are presented for the performance of error control codes. Many of them are tighter than the ones presented above.

As seen from the results, it should be remembered that the soft decoding has a power advantage of 1-2 dB compared to hard decoding.
Example 2: (7,4) Hamming-code

\[ \Pr[\text{block error}] = 1 - (1 - p)^7 - 7p(1 - p)^6 \]

\[ p = 0.01 \Rightarrow \Pr[\text{block error}] = 0.002 \]

In the case of AWGN-channel \( \sigma_c^2 = \frac{7}{4} \sigma_u^2 \), and the BSC error probability is larger than in the uncoded system with the same input levels.

The power advantage due to this kind of coding is only a fraction of dB.
Parity Check Matrix

As it was said, soft and hard ML decoders find the closest codewords (in Euclidean or Hamming distance) to the received block. For large $k$ and $n$ this becomes difficult (there are $2^k$ distances to compare).

For soft decoding, there are no other practical approaches, and thus they are usually impractical for large block sizes. Soft decoding is used mostly with convolution codes.

For hard decoding, there are some more efficient techniques. The idea is to use codes with certain algebraic structure which can be exploited in the decoding. Here just the basic ideas are illustrated, a more complete discussion of error control coding from algebraic point of view is available in the course Coding Theory offered by the Institute of Mathematics.

The generator matrix of a systematic code $(n,k)$ has the following structure

$$G = [I_k \mid P]$$

The codeword can be obtained as:

$$c = bG = [b \ a]$$

where $b$ is source $k$-bit vector to be coded
$a$ are $n - k$ parity check bits

The following can be shown:

$$a = bP \Rightarrow bP \oplus a = 0$$

$$[ba] \begin{bmatrix} P \\ I_{n-k} \end{bmatrix} = 0$$

$$cH' = 0, \quad H = [P \mid I_{n-k}]$$

The matrix $H$ is called parity check matrix.
Example

Example 1: \((k+1, k)\) parity check code.

The parity check matrix is: \(H = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}\)

Example 2: \((7, 4)\) Hamming-code

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

For example \(c_1 = [0, 1, 1, 0, 0, 0, 1]\) is codeword because: \(c_1H^T = 0\).

The vector \(c_2 = [0, 1, 1, 0, 1, 0, 1]\) is not codeword because: \(c_2H^T = [1, 0, 0]\).

The parity check matrix can be used for checking whether a bit pattern is an existing codeword.
Syndrome

For a systematic code, the parity check matrix is easy to construct.

It is possible to define the parity check matrix for any linear block code.

For a received vector, the parity check matrix can be used to avoid calculation of the distance to every codeword.

A received bit vector can be written as

\[ \bar{c} = c \oplus e \]

where, \( e \) is the error pattern.

The syndrome is defined as:

\[ s = \bar{c}H' = cH' + eH' = eH' \]

The syndrome is zero if the received vector is a codeword. Efficient decoders use the syndrome to flag the position of the error, which can then be corrected.
Hamming Codes

For any positive integer $m$ there is a Hamming code with

$$(n, k) = (2^m - 1, 2^m - 1 - m).$$

The parity check matrix has dimension $n \times (n - k) = n \times m$. For a Hamming code, the parity check matrix is constructed by letting the $n$ columns to be all the possible binary vectors with $n - k$ elements, except for the zero vector.

Once the parity check matrix is found, the generator matrix can be found easily.

Any Hamming code has the minimum distance $d_{H,\text{min}} = 3$, thus all Hamming codes can correct a single error ($t = 1$)

Hamming codes are perfect codes.
Cyclic Codes

The cyclic codes have a rich algebraic properties, which lead to efficient decoding algorithms. They are the most practical block code class.

A \((n, k)\) linear block code is cyclic if any cyclic shift of a codeword produces another codeword.

Some Hamming codes are cyclic codes.

**Example 2:** The \((7, 4)\) Hamming code: The cyclic shifts of 1000101 produce all codewords:

\[
\begin{align*}
1100010 & \quad 0110001 & \quad 1011000 & \quad 0101100 & \quad 0010110 \\
0001011 & \quad 1000101
\end{align*}
\]

**BCH-codes** (Bose, Chaudhuri, Hocquenghem)

For any positive integers \(m\) and \(t\), there is a \(t\)-error correcting BCH code with

\[
n = 2^m - 1 \quad k \geq n - mt
\]

BCH codes are an important class of multiple-error correcting codes used mainly for hard decoding systems. Many efficient decoding techniques have been found.

**Reed-Solomon codes** (RS-codes) are an important subset of BCH codes. They are important because there are many efficient practical decoding techniques. The RS codes are able to correct bursts of errors.
Cyclic codes

1. The sum of any two code words is a code word (linearity)
2. The result of cyclically shifting a code word is another code word.

A subset of cyclic codes have a generator matrix $G$ whose rows are shifted versions of each other. An example of such a code is

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{pmatrix}$$

This kind of code can be expressed by, for example, the last row of $G$ alone.

Generator polynomial

$$a = [ a_{N-1} \ a_{N-2} \ \cdots \ \ a_0 ]$$

binary case: $a_i = 0$ or 1

polynomial representation

$$a(D) = \sum_{i=0}^{N-1} a_i D^i$$

For example:

$$g_3(D) = D^3 + D^2 + 1$$
If we denote \( g_3(D) = g(D) \) then
\[
\begin{align*}
g_2(D) &= Dg(D) \\
g_1(D) &= D^2g(D) \\
g_0(D) &= D^3g(D)
\end{align*}
\]

The coding operation \( x = uG \) can now be represented in terms of polynomial multiplication \( x(D) = u(D)g(D) \).

Here, \( g(D) \) is the generator polynomial of this code.

Example: Assume data \( u = [1 \ 0 \ 1 \ 1] \). That is \( u(D) = D^3 + D + 1 \).

With the matrix operation we get:
\[
x = uG' = [1 \ 0 \ 1 \ 1] \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
\]

With the polynomial operation we get
\[
x(D) = u(D)g(D)
\]
\[
= (D^3 + D + 1)(D^3 + D^2 + 1)
\]
\[
= D^6 + D^5 + D^4 + D^3 + D^2 + D + 1
\]

But how about the cyclic shift property?? Our example was a (7,4) cyclic Hamming code. So, the code words should be 7 bits long, and for example, if \( x = [1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1] \) is a code word then \( x' = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1] \) also is a code word.
In the polynomial representation, a shift is basically multiplication by $D$ but
\[
x(D) = D^6 + D^5 + D^2 + 1 \quad \text{and} \quad Dx(D) = D^7 + D^6 + D^3 + D \]
which is 8 bits long.

By equating $D^7$ to 1, we would get the correct $x'(D) = D^6 + D^3 + D + 1$. In polynomial algebra this may be expressed as
\[
x'(D) = Dx(D) \mod (D^7 + 1)
\]

In other words, divide $Dx(D)$ by $(D^7 + 1)$ and the remainder is $x'(D)$.

The same thing by long division:

\[
\begin{array}{cccc}
D^7 & +1 & \\
\hline
D^7 & D^6 & +D^3 & +D \\
D^7 & D^6 & +D^3 & +D & +1 \\
\hline
\end{array}
\]

Properties of the generator polynomial

1. $g(D)$ is the unique nonzero code polynomial of least degree in the code.
2. $g(D)$ is of the form $D^{N-K} + g_{N-K-1}D^{N-K-1} + \ldots + g_1D + 1$.
3. Every code word is a multiple of $g(D)$, and there are $2^K$ distinct code words in the code.
4. $g(D)$ is a factor of $D^N - 1$ (or $D^N + 1$).
5. $g(D)$ generates an $(N,K)$ cyclic code.
Systematic Cyclic Codes

Systematic cyclic codes

Direct application of the polynomial multiplication does not, in general, produce a systematic code. Systematic codes can be produced with the following steps:

1. Shift the information polynomial \( u(D) \) left by \( N-K \) places; that is, multiply \( u(D) \) by \( D^{N-K} \).
2. Divide \( D^{N-K}u(D) \) by \( g(D) \) to obtain the remainder \( D^{N-K}u(D) \mod g(D) \).
3. Add the remainder to \( D^{N-K}u(D) \) to obtain the code polynomial; that is, \( x(D) = D^{N-K}u(D) + D^{N-K}u(D) \mod g(D) \).

This form is also assumed in the next discussion.

The same idea can be implemented in the following way (example \( g(D) = D^3 + D^2 + 1 \)):

1. With switch S1 in position 1 and S2 closed, shift the information bits into the shift register and also on to the channel.
2. Once all the information bits are shifted into the shift register in \( K \) shifts, now with switch S1 in position 2 and S2 open, the contents of the \((N-K)\)-stage shift register are shifted onto the channel.
**The Syndrome of Cyclic Codes**

The syndrome of cyclic codes

Earlier we defined the syndrome $S$ of a block code as

$$S = yH^T$$

If the received word $y$ is a code word, then $S = 0$ and a nonzero syndrome indicates that one or more errors occurred on the channel.

In polynomial representation, every code word (polynomial) is divisible by $g(D)$. Let $y(D)$ correspond to the received word, and $q(D)$ be the quotient obtained by dividing $y(D)$ by $g(D)$ and $s(D)$ be the remainder. Then

$$y(D) = q(D)g(D) + s(D)$$

where the syndrome (polynomial) $s(D)$ is zero if the received word is a code word, and nonzero polynomial of degree at most $N-K-1$ if errors occurred in the channel. The syndrome can be obtained with the following division circuit:

![Division Circuit Diagram]

Again we have the example case $g(D) = D^3 + D^2 + 1$. With the received word $y = [1 0 0 1 1 1]$, we get the following states of the circuit:

<table>
<thead>
<tr>
<th>Shift</th>
<th>Input</th>
<th>$D^2$</th>
<th>$D^1$</th>
<th>$D^0$</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The last state of the shift register corresponds to the syndrome $s(D)=D^2$. This may be verified by dividing $y(D)$ by $g(D)$. 
The book mentioned on the first page gives examples of good codes.

As examples of practical codes (both block and convolutional), we present the codes used in GSM mobile communication system. First a summary:

<table>
<thead>
<tr>
<th>Channel</th>
<th>Input rate, kbit/s</th>
<th>Input block</th>
<th>Coding</th>
<th>Output block</th>
<th>Interleaving</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCH/FS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>La</td>
<td>13</td>
<td>50</td>
<td>Parity (3 bits) Convolutional 1/2</td>
<td>456</td>
<td>On 8 half bursts</td>
</tr>
<tr>
<td>lb</td>
<td></td>
<td>132</td>
<td>Convolutional 1/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II</td>
<td></td>
<td>78</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCH/FH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>la</td>
<td>5.6</td>
<td>22</td>
<td>Parity (3 bits) Convolutional 1/2</td>
<td>228</td>
<td>On 4 half bursts</td>
</tr>
<tr>
<td>lb</td>
<td></td>
<td>73</td>
<td>Convolutional 1/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II</td>
<td></td>
<td>17</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCH/F9.6</td>
<td>12</td>
<td>240</td>
<td>Convolutional 1/2</td>
<td>456</td>
<td>On 22 unequal burst portions</td>
</tr>
<tr>
<td>TCH/H4.8</td>
<td>6</td>
<td>240</td>
<td>punctured 1/15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCH/F4.8</td>
<td>6</td>
<td>120</td>
<td>Add 32 null bits Convolutional 1/3</td>
<td>456</td>
<td>On 22 unequal burst portions</td>
</tr>
<tr>
<td>TCH/F2.4</td>
<td>3.6</td>
<td>72</td>
<td>Convolutional 1/6</td>
<td>456</td>
<td>On 8 half bursts</td>
</tr>
<tr>
<td>TCH/H2.4</td>
<td>3.6</td>
<td>144</td>
<td>Convolutional 1/3</td>
<td>456</td>
<td>On 22 unequal burst portions</td>
</tr>
<tr>
<td>SCH</td>
<td></td>
<td>25</td>
<td>Parity (10 bits) Convolutional 1/2</td>
<td>78</td>
<td>On 1 S burst</td>
</tr>
<tr>
<td>RACH</td>
<td>8</td>
<td>8</td>
<td>Parity (6 bits) Convolutional 1/2</td>
<td>36</td>
<td>On 1 access burst</td>
</tr>
<tr>
<td>FACCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCH/8, SACCH, BCCH, PAGCH</td>
<td>184</td>
<td>Fire code 224/184 Convolutional 1/2</td>
<td>456</td>
<td>On 8 half bursts</td>
<td></td>
</tr>
</tbody>
</table>
Reed-Solomon Codes

Reed-Solomon codes (RS codes) are defined for codeword lengths

\[ n = 2^m - 1 \]

where \( m \) is a positive integer. Each codeword consists of \( n \) symbols of \( m \)-bits.

An \((n,k)\) Reed-Solomon code is able to correct

\[ t = (n-k)/2 \]

erroneous symbols in each codeword. It doesn’t make any difference whether a single bit or all the bits are wrong.

**Example:** \((255,235)\)-Reed-Solomon code is a commonly used basic code, e.g., in the family of DVB digital TV transmission systems.

- codeword length is \( 255 \times 8 = 2040 \) bits
- each codeword carries 1880 information bits, consisting of 235 symbols of 8 bits.
- the code is able to correct 10 erroneous symbols in each codeword, i.e., maximally 80 erroneous bits (in case all the bits of all erroneous symbols are wrong)
- the code can correct any error burst of 72 consecutive bits within a codeword
In Reed-Solomon codes, the possible codeword lengths in bits are: 3x2, 7x3, 15x4, 31x5, 63x6, 127x7, 255x8, ...

So there are only a rather limited sets of alternative lengths.

More flexibility can be achieved by considering the so-called shortened Reed-Solomon codes (actually, the principle can be applied also for other block codes).

**Example:** Using the basic (255,235) Reed-Solomon code, a shortened code (208, 188) can be derived as follows:

1. In the coder, 235-symbol words are generated by inserting 47 0-symbols into the sequence of 188 symbols to be coded.
2. These words are coded by using a coder for the basic (255, 235) RS code.
3. The extra 47 zero symbols are removed from each codeword (their places are fixed and known in advance). The remaining 208 symbols are transmitted.
4. In the decoder, the zero symbols are again inserted to the proper positions, then the decoding is done using the decoder of the basic code, and the 188 information symbols are collected from the decoded sequence.
Figure 3.1.2. Comparison of RS codes with the same rate

Figure 3.1.4. Coding gain with 16-QAM
A convolutional coder is a finite memory system, different from block coders which are memoryless. Convolutional codes are used more often than block codes, because they are conceptually and practically simpler, and their performance is better than the performance of good block codes. The performance is better mainly thanks to practical soft decoding techniques.

The main parameters:
- Coding rate $R = \frac{k}{n}$.
- Constraint length $M$ (in some sources $kM$).

Convolutional codes are constructed using modulo two summations and delay elements.

The convolutional coder is working continuously:
- Coder takes $k$ input bits during each cycle.
- It produces $n$ output bits during each cycle.
- Output bits are determined by using $M$ previous source bits.

The code is systematic if all the $k$ bits are passed to the output unchanged.

Convolutional codes can be described by using generator and parity check matrices, in the same way as block codes. In this case, these matrices have polynomials as entries, different from block codes where the entries were 0's and 1's.

It is also possible to represent the input bit sequence as polynomial in $D$ ($D$ is in place of $z^{-1}$). This is a modulo-two z-transform representation:

Example: Bit sequence 1 0 0 1 1 0 0 1 has the "z-transform"

$$1 + D^3 + D^4 + D^7$$
Generator Matrix and Parity Check Matrix

The coder is defined by the following z-transform relation

\[ C(D) = B(D)G(D) \]

where \( G(D) \) is the generator matrix and

\[
\begin{align*}
B(D) &= \begin{bmatrix} B^{(1)}(D), B^{(2)}(D), \ldots, B^{(k)}(D) \end{bmatrix} \\
C(D) &= \begin{bmatrix} C^{(1)}(D), C^{(2)}(D), \ldots, C^{(n)}(D) \end{bmatrix}
\end{align*}
\]

Here \( B^{(i)}(D) \) and \( C^{(i)}(D) \) are z-transforms of input and output bit sequences.

The highest degree of \( G(D) \) is \( M-1 \), where \( M \) is the constraint length of the code.

If the code is systematic, the parity check and generator matrices are related, and they can be determined from each other.

In the following examples, we can see how the generator and parity check matrices can be used in the same way as for block codes.

We will also see an example of how non-systematic code can be modified to become systematic.
**Example 1**

Code rate $R = k/n = 2/3$

Constraint length $M = 2$

Generator matrix:

\[ G(D) = \begin{bmatrix} 1 & 0 & 1 \oplus D \\ 0 & 1 & D \end{bmatrix} \]

The code is systematic. The parity check matrix is:

\[ H(D) = [1 \oplus D, \ D, \ 1] \]

Let us now check if this is really a parity check matrix:

\[ C(D)H'(D) = \begin{bmatrix} B^{(1)}(D), B^{(2)}(D), (1 \oplus D)B^{(1)}(D) \oplus DB^{(2)}(D) \end{bmatrix} \begin{bmatrix} 1 \oplus D \\ D \\ 1 \end{bmatrix} = (1 \oplus D)B^{(1)}(D) \oplus DB^{(2)}(D) \oplus (1 \oplus D)B^{(1)}(D) \oplus DB^{(2)}(D) = 0 \]

This is valid for all codewords.

The parity check matrix is very helpful for systematic codes. In this case it is possible to set $C^{(1)}(D) = B^{(1)}(D)$, $C^{(2)}(D) = B^{(2)}(D)$, and $C^{(3)}(D)$ is determined using the parity check matrix:

\[ C(D)H'(D) = (1 \oplus D)C^{(1)}(D) \oplus DC^{(2)}(D) \oplus C^{(3)}(D) = 0 \]

\[ \Rightarrow C^{(3)}(D) = (1 \oplus D)C^{(1)}(D) \oplus DC^{(2)}(D) \]
Example II

Coder rate is $R = k/n = 1/2$.
Constraint length $M = 3$.

Generator matrix:

$$G(D) = \begin{bmatrix} 1 \oplus D^2, & 1 \oplus D \oplus D^2 \end{bmatrix}$$

The code is not systematic.

Instead of parity check matrix, the following modulo two algebra can be used:

$$C^{(1)}(D) = (1 \oplus D^2)B(D)$$
$$C^{(2)}(D) = (1 \oplus D \oplus D^2)B(D)$$

If both sides of the first equation are multiplied with $(1 \oplus D \oplus D^2)$ and both sides of second equation with $(1 \oplus D^2)$, then the left sides of these two equations are equal, and we can write

$$(1 \oplus D \oplus D^2)C^{(1)}(D) = (1 \oplus D^2)C^{(2)}(D)$$

or

$$(1 \oplus D \oplus D^2)C^{(1)}(D) \oplus (1 \oplus D^2)C^{(2)}(D) = 0$$

Thus we can define a parity check matrix as

$$H(D) = \begin{bmatrix} (1 \oplus D \oplus D^2), & (1 \oplus D^2) \end{bmatrix}.$$
Example II (cont.)

Given a parity check matrix, it is easy to design a systematic coder. First, we can write:

\[ C(D)H'(D) = (1 \oplus D \oplus D^2)C^{(1)}(D) \oplus (1 \oplus D^2)C^{(2)}(D) = 0 \]

To get a systematic code, assign \( C^{(1)}(D) = B(D) \). Now we can write:

\[ (1 \oplus D \oplus D^2)B(D) = (1 \oplus D^2)C^{(2)}(D) \]

Based on this equation, it is possible to construct a systematic coder, as shown in figure (a). This system is a cascade of two linear subsystem whose order can be reversed to get the implementation in (b). (This is procedure used in design in recursive digital filters.)

This coder and the initial coder have the same parity check matrix. Further, in all important respects these two codes are equivalent.
Decoding

We consider here both hard decoding and soft decoding of convolutional codes. In general, soft decoding gives better performance.

In the case of convolutional codes, it is very natural to perform ML sequence detection. The decoder chooses the sequence that is closest to the observation in Hamming or Euclidean distance.

It is convenient to represent convolutional codes as Markov chains, as convolutional coding is a shift register process.

The state transition diagram of a code can be represented by using the trellis diagram. The trellis diagram shows the progression through states over time. The state of the code (Markov chain) corresponds to the bits stored in the delay elements in the coder.

The decoder complexity depends essentially on the number of states in the diagram, which is determined by the total number of shift-register stages in the coder. The number of states is less than or equal to $2^{k(M-1)}$.

The decoder chooses the shortest path based on the received sequence.

**Example I:** Below one stage of the trellis diagram of the example code with $R=2/3$ is shown. The number of states is 4 and the trellis is fully connected.
Example II

In this case there are two bits in the delay line of the coder. The number of states is 4. Below different versions of the trellis diagram are shown.

Figure 11-11. a. A four-state trellis illustrating all possible state transitions of the Markov chain in figure 7-13, which represents conv(1/2), assuming the initial and termination states are zero. b. One stage of the trellis is shown with the transitions labeled with \((B_k, C_k^{(1)}, C_k^{(2)})\), where \(B_k\) is the input bit that triggers the transition and \(C_k^{(1)}\) and \(C_k^{(2)}\) are the outputs produced. c. The branches of the trellis labeled with the transmitted symbols rather than the bits out of the coder for binary antipodal signaling with levels \(\pm a\).
This is a coded transmission system with a soft decoder. We assume additive white Gaussian noise and binary antipodal signalling, in which case Viterbi algorithm can be used for performing optimal ML-sequence detection.

These are the important parameters for coded and uncoded signalling:

<table>
<thead>
<tr>
<th></th>
<th>uncoded</th>
<th>coded</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel symbols</td>
<td>±(a_u)</td>
<td>±(a_c)</td>
</tr>
<tr>
<td>noise variance</td>
<td>(\sigma_u^2)</td>
<td>(\sigma_c^2)</td>
</tr>
</tbody>
</table>

In the following, we compare the performance of uncoded and coded systems, based on soft and hard decoding. The analysis is done for the convolutional code with rate \(\frac{1}{2}\), that was introduced earlier.

For uncoded system, the probability of symbol error is equal to probability of bit error:

\[
\Pr[\text{bit error}] = \Pr[\text{symbol error}] = Q\left(\frac{a_u}{\sigma_u}\right)
\]
**Example I: Soft Decoding**

In the following we use the notation:

- Noisy observations:
  \[ y_k, y_{k+1}, \ldots \]

- Detected symbols:
  \[ \hat{a}_k, \hat{a}_{k+1}, \ldots \]

One stage of the trellis diagram with all possible transitions is given here. It is used for decoding of one single bit. The ML detector computes the branch metrics as

\[ d^2 = (y_k - \hat{a}_k)^2 + (y_{k+1} - \hat{a}_{k+1})^2 \]

The ML- decoder chooses the path with minimum path metrics. For soft decoding, the Euclidean metric is used.

Probability of error depends on the error event with minimum distance. For this example case, the minimum distance error path (critical path) is shown in the following figure.

The square of the Euclidean distance between the correct path and the error event is:

\[ d_{\text{min}}^2 = 20a_c^2 \]

In this case, it is easy to check that there are no distinct paths that are closer to each other. Furthermore, due to the linearity of the code, there are similar error events in every possible correct path.
Example I: Soft Decoding (cont.)

The probability of symbol error can be expressed as:

\[
\Pr[\text{symbol error}] = KQ\left(\frac{d_{\text{min}}}{2\sigma}\right)
\]

In the example case, the symbol error is same as bit error, thus:

\[
\Pr[\text{bit error}] = \Pr[\text{symbol error}] = Q\left(\frac{\sqrt{20}a_c}{2\sigma_c}\right) = Q\left(\frac{\sqrt{5}a_c}{\sigma_c}\right)
\]

Here \(d_{\text{min}}\) is minimum Euclidean distance of an error event, and \(K\) is a constant. In general, finding \(K\) and \(d_{\text{min}}\) can be very difficult. If the code is linear, this is simplified by using Viterbi algorithm in a modified form.

If the transmission rate is same, in coded system the signal bandwidth is doubled. This means that the noise power in coded system is twice the noise power of uncoded system, i.e.

\[
\sigma_c = \sqrt{2}\sigma_u
\]

If it is desired to achieve the same performance with coded and uncoded systems, then the argument of Q function should be the same, in other words

\[
5\text{SNR}_c = \text{SNR}_u
\]

If we take into account increased noise level in the coded system then:

\[
\frac{a_u}{\sigma_u} = \frac{\sqrt{5}a_c}{\sigma_c} = \frac{\sqrt{5}a_c}{\sqrt{2}\sigma_u}
\]

So the true coding gain is \(10\log_{10}(5/2) = 4\, \text{dB}\)
Performance of Hard Decoding: Example I

In the following, the hard decoding is compared to both soft decoding and uncoded system. For hard decoder, the channel can be modelled as a BSC.

The analysis is similar to analysis for soft decoding. The main differences are:
- The Hamming distance is used
- Instead of error function $Q()$, the binomial $Q(, )$-function is used to calculate probability of error.

A critical error path is given below:

Here, the correct bit sequence is: (0,0) (0,0) (0,0)
The error sequence is: (1,1) (0,1) (1,1)

The corresponding Hamming distance is 5, and the probability of error becomes:

$$Pr[\text{bit error}] = Pr[\text{detection error}] = Q(5, p) = 10p^3(1-p)^2 + 5p^4(1-p) + p^5$$

For small value of error probability in BSC, this is approximated by:

$$Pr[\text{bit error}] = 10p^3$$
Performance of Hard Decoding: Example I (cont.)

For comparison, we assume the same probability of error for coded and uncoded systems, e.g., $10^{-5}$. Then it is possible to derive $p$ for the BSC model of coded system:

$$10p^3 = 10^{-5} \implies p = 0.01$$

The signalling is antipodal, therefore:

$$Q(a_h / \sigma_c) = Q(a_h / \sqrt{2} \sigma_u) = 0.01 \implies a_h = 3.4 \sigma_u$$

In the uncoded system, the same probability of error is achieved when $a_u = 4.3 \sigma_u$:

Thus the coding gain becomes: $20 \log_{10}(4.3 / 3.4) \approx 2$ dB

This is smaller compared to case of soft decoding (it was 4 dB)
More about Convolutional Codes

The concept of free distance, $d_{\text{free}}$, has the same significance for convolutional codes as Hamming distance has in case of block codes. It gives an idea about the performance of the code.

As an example, for the code of page 371, the free distance is 5.

Good convolutional codes have been found through computer search methods. The following table gives some of those. The elements of the generator matrix are given there in octal form. As an example, the first code of the table is one of the widely used examples in the Lee&Messerschmitt book (see page 371).

![Table showing some convolutional codes](image)

Punctured Convolution Codes

Puncturing means that some of the bits from the output of a normal convolutional coder are removed, in a regular pattern. This is done, of course, in such way that the data can be perfectly decoded in case of ideal channel.

In puncturing, the code-rate is increased, but the free distance is reduced.

The following table lists some punctured codes for rates 2/3 and 3/4. In both cases, the basic code is the rate 1/2 convolution code of page 371.

<table>
<thead>
<tr>
<th>$d_{free}$</th>
<th>$\begin{array}{c} \text{siirtomatriisi} \ \hline 3 \ 4 \ 5 \ 6 \ 6 \ 8 \ 8 \end{array}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,5,7,X</td>
<td>15,13,15,X</td>
</tr>
<tr>
<td>31,33,31,X</td>
<td>73,41,73,X</td>
</tr>
<tr>
<td>163,135,163,X</td>
<td></td>
</tr>
<tr>
<td>337,251,337,X</td>
<td></td>
</tr>
<tr>
<td>661,473,661,X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d_{free}$</th>
<th>$\begin{array}{c} \text{siirtomatriisi} \ \hline 3 \ 4 \ 4 \ 5 \ 6 \ 6 \ 6 \end{array}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,7,5,X,X,7</td>
<td>15,17,15,X,X,17</td>
</tr>
<tr>
<td>25,37,X,37,X,37</td>
<td></td>
</tr>
<tr>
<td>61,53,X,53,X,53</td>
<td></td>
</tr>
<tr>
<td>135,163,X,163,X,163</td>
<td></td>
</tr>
<tr>
<td>205,307,X,307,X,307</td>
<td></td>
</tr>
<tr>
<td>515,737,X,737,X,737</td>
<td></td>
</tr>
</tbody>
</table>
**Example:** The code with transfer matrix \((7,5,7,x)\) is obtained from the basic code of page 371 (see below) by forgetting every other \(C_k^I\) output.

The code with transfer matrix \((5,7,5,x,x,7)\) is obtained from the basic code of page 371 (see below) by sending the following bit-stream:

\[
C_1^{(1)}, C_1^{(2)}, C_2^{(1)}, C_2^{(2)}, C_3^{(1)}, C_3^{(2)}, C_4^{(1)}, C_4^{(2)}, C_5^{(1)}, C_5^{(2)}, C_6^{(2)}, C_7^{(1)}, \ldots
\]

One very interesting feature of puncturing is that several codes with different rates can be obtained form a single basic code.

This makes it possible to develop systems, where the useful bit-rate and error correction performance can be traded-off in a flexible way. Such approach is used in the new European digital TV systems, DVB-T and DVB-S.
Notice that here $K$ is the constrained length.

### TABLE 6.2 Coding Gain Upper Bounds for Some Convolutional Codes

<table>
<thead>
<tr>
<th>Rate $\frac{1}{2}$ codes</th>
<th>Rate $\frac{2}{3}$ codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>$d_f$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>


---

### 7.3LE 6.1 Comparison of Systematic and Nonsystematic Free Distance, Rate $\frac{2}{3}$

<table>
<thead>
<tr>
<th>Constraint length</th>
<th>Free distance systematic</th>
<th>Free distance nonsystematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>


---

**Figure 6.19** Bit error probability versus $E_b/N_0$ for rate $\frac{2}{3}$ codes using coherent BPSK over a BSC, Viterbi decoding, and a 32-bit path memory. (Reprinted with permission from J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.*, vol. COM-19, no. 7, October 1971, Fig. 7, p. 84. © 1971 IEEE.)
TABLE 6.3 Basic Coding Gain (dB) for Soft Decision Viterbi Decoding

<table>
<thead>
<tr>
<th>Uncoded $E_b/N_0$ (dB)</th>
<th>Code rate</th>
<th>$\frac{1}{7}$</th>
<th>$\frac{1}{5}$</th>
<th>$\frac{1}{6}$</th>
<th>$\frac{1}{7}$</th>
<th>$\frac{1}{6}$</th>
<th>$\frac{1}{7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_a$</td>
<td>$K$</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>6.8</td>
<td>$10^{-3}$</td>
<td>4.2</td>
<td>4.4</td>
<td>3.3</td>
<td>3.5</td>
<td>3.8</td>
<td>2.9</td>
</tr>
<tr>
<td>9.6</td>
<td>$10^{-5}$</td>
<td>5.7</td>
<td>5.9</td>
<td>4.3</td>
<td>4.6</td>
<td>5.1</td>
<td>4.2</td>
</tr>
<tr>
<td>11.3</td>
<td>$10^{-7}$</td>
<td>6.2</td>
<td>6.5</td>
<td>4.9</td>
<td>5.3</td>
<td>5.8</td>
<td>4.7</td>
</tr>
<tr>
<td>Upper bound</td>
<td></td>
<td>7.0</td>
<td>7.3</td>
<td>5.4</td>
<td>6.0</td>
<td>7.0</td>
<td>5.2</td>
</tr>
</tbody>
</table>


Figure 6.18 Bit error probability versus $E_b/N_0$ for rate $\frac{1}{7}$ codes using coherent BPSK over a soft quantized channel, Viterbi decoding, and a 32-bit path memory. (Reprinted with permission from J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.*, vol. COM19, no. 5, October 1971, Fig. 5, p. 84, © 1971 IEEE.)
Concatenated Coding

A commonly used solution in digital transmission systems (e.g., DVB systems) includes two different error correcting codes, and also interleaving in between:

This has turned out to be a practical way to design error control coding systems that provide a very low BER for the application, using a rather low SNR (or high ‘raw BER’ before the channel decoder). Typically, the ‘raw BER’ is 0.1...0.01, the BER after convolution decoding $10^{-3}$...$10^{-4}$, and the BER after block decoder $10^{-9}$...$10^{-12}$.

On the other hand, such solutions with rather high ‘raw BER’ make it possible to get closer to channel capacity than using higher SNR.
The inner code is typically a convolution code (or trellis code or, in latest developments, a turbo code, an advanced structure based also on convolutional coding ideas). Using convolution type of code makes it possible to get the benefit of soft decoding, which is very important when the ‘raw BER’ is high. Using the puncturing ideas, it is also possible to make flexible system, where the code rate and the SNR requirements can be adapted according to the channel conditions and application requirements.

The outer code is typically a block code, like Reed-Solomon.

The channel and the inner decoder produce often error bursts, and by using the (de)interleaver, the burst errors are spread out evenly in the symbol stream going to the outer coder. This helps the outer decoder in its tasks and thus helps to reduce the final bit error rate.
Interleaving

In interleaving, the symbol/bit stream is reordered deterministically. The goal is to move the consecutive bits coming from the channel as far as possible from each other. This removes the correlations in the bit/symbol-stream.

The lengths of the interleavers to be used in practice are limited by:

- required memory needed for the interleaver
- induced processing delay.

Interleaving is commonly used in the context of concatenated coding, but it can also be used otherwise to reduce the correlations in the bit/symbol streams.

There are two commonly used principles for interleaving:

- **Block interleaving**: The incoming symbol stream is written into an \( M \times N \) table column-wise and read out row-wise. The de-interleaver is, of course, implemented in the opposite way.

- **Convolution interleaving**: This is a slightly more complicated structure, saving about half of the needed memory:
In the case of block and convolutional codes, the redundancy was introduced by adding some symbols in the symbol stream. Thus, the symbol rate and required bandwidth are increased. This will increase the noise in the system. Therefore, the designed code must have larger coding gain to compensate this.

It is possible to introduce the redundancy by using larger constellation, i.e., by using more symbols in the alphabet. In this case the symbol rate is not changed. This technique is useful for band limited signals. The resulting code is sometimes called signal-space code.

The symbols in the new constellation are more closely spaced, and the noise immunity is reduced. The coding gain must be high enough to compensate for this.

The input constellation size is $2^k$, with $k$ bits per symbol, and after coding the size of constellation is $2^n$. Thus a code with code rate $k/n$ is needed.

The code to introduce redundancy can be a block code or a convolutional code. When a convolutional code is used to get a signal-space code, the result is called a trellis code.

Source: Lee&Messerschmit, Chapter 12.
Assume that we have an AWGN channel, and an acceptable data rate is achieved with constellation size of $M$ and probability of error is achieved without coding at some SNR.

Using coding it is possible to:
- Reduce the error probability by keeping the same SNR, or
- Achieve the same error probability with smaller SNR.

The improvements in performance are limited by the Shannon channel capacity (see pages 26 and 27). Below, we show the key observation that the most of this theoretical maximum improvement can be achieved using a constellation of size $2M$ and suitable code.

**Example:**
It is desired to have rate of 4 bits per symbol and probability of error not larger than $10^{-5}$.
- 16-QAM case: required SNR is 19 dB
- 32-QAM, theoretical lower bound for SNR is 13 dB
  - obtained coding gain is 6dB
- Shannon bound is 11.7 dB
  - With larger constellations, the coding gain could be improved by 1.3 dB, at most.

The coding gain of 3-4 dB can usually be achieved easily.

Trellis coding is commonly used in various applications, like voice band modems.
The performance of trellis codes depends on the number of states in trellis diagram of the used code. The following table gives some idea about the achievable performance.

<table>
<thead>
<tr>
<th>Number of states</th>
<th>Coding gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 dB</td>
</tr>
<tr>
<td>8</td>
<td>4 dB</td>
</tr>
<tr>
<td>16</td>
<td>close to 5 dB</td>
</tr>
<tr>
<td>128</td>
<td>close to 6 dB</td>
</tr>
</tbody>
</table>

By their nature, trellis codes lend themselves to soft decoding using the Viterbi algorithm. At reasonably high SNR, the symbol error probability is given by:

$$\Pr[\text{symbol error}] = K Q \left[ \frac{d_{E,\min}}{2\sigma} \right]$$

Here $K$ is a constant (between lower and upper limits given in the book by Eqs. (7.95) and (7.87)). In simple example cases, the value of $K$ can be determined easily. However, in many cases the constant is approximated $K=1$.

The performance of trellis codes is dominated by the error event with minimum Euclidean distance. In general, trellis codes are not linear even when the underlying convolutional code is linear.

In this case it is not safe to simplify the calculation of minimum distance by finding the error event closest to the zero state trajectory. Instead, one has to find systematically the minimum distance error event for each possible correct path through the trellis, e.g., using a modification of the Viterbi-algorithm.
Example of Basic Trellis Codes

In the following, a simple trellis code is presented. It uses convolutional code with rate $R = 1/2$, 4-PSK modulation and 1 bit per symbol.

The convolutional coder (without the line coder) has been considered earlier in this material.
Performance of Basic Trellis Codes

In the previous slide, we have seen the minimum distance error event assuming the correct state trajectory is zero. The distance is calculated using complex arithmetic:

\[ d_{E,\text{min}} = \sqrt{|a + a|^2 + |a - ja|^2 + |a + a|^2} = \sqrt{10}a \]

It is easy to observe that this is error event with minimum distance, based on the following facts:
- all branches diverging from the same node have distance \(2a\) from each other
- all branches converging on the same node have distance \(2a\)
- after two parts diverge, all possible combinations of subsequent branches have distance \(\sqrt{2a}\)

Thus, the minim distance is:

\[ d_{E,\text{min}} \geq \sqrt{(2a)^2 + 2a^2 + (2a)^2} = \sqrt{10}a \]

The probability of error is obtained as:

\[ \Pr[\text{biterror}] \approx \Pr[\text{symbolerror}] \approx Q\left(\frac{\sqrt{10}a}{2\sigma}\right) \]

In the example case, every state trajectory, corresponding to 3 symbols, has exactly one error event at minimum distance, and this error event has exactly one symbol error, so \(K=1\).

To compare this performance to an uncoded system, we first observe that the noise variance is the same in both cases because the signal bandwidth is the same. An uncoded 2-PSK system has the same transmit power as coded 4-PSK, and carries the same number of source bits. The probability of error is:

\[ \Pr[\text{biterror}] = Q\left(\frac{a}{\sigma}\right) \]

Thus the coding gain is: \(20\log(\sqrt{10} / 2) = 4 \text{ dB}\)

The same improvement is achieved by convolutional coding with soft decoding, but in the case of trellis coding without increasing the bandwidth. Trellis coding is usually preferred over convolutional or block coding for bandlimited channels, unless nonlinearities or hardware complexity make them impractical.
More Elaborate Trellis Codes

In the above example, one source bit was processed to give two coded output bits. Two coded bits are used to produce alphabet of size four. The technique can be easily extended to make use of alphabet larger than four.

In this figure, the trellis code is extended by using one uncoded bit and a larger alphabet (size 8 in this case), thus the number of source bits per symbol is increased.

The transition from one state to another is controlled entirely by the source bit that goes to the coder. Each transition from one state to another occurs for two possible values of uncoded source bit, zero and one. This can be represented by showing two parallel transitions between every pair of states in trellis diagram. The trellis diagram of the figure below corresponds to example of the figure above.
This technique can be generalized as shown here.

First, based on bandwidth limitations, determine the symbol rate that can be transmitted. Next, determine the size of alphabet $2^m$ that would be required (without coding) to transmit the source bits at the desired rate. Then double the size of alphabet to $2^{m+1}$ and introduce a channel coder that produces the extra bit. The coder does not need to code all incoming bits, however leaving some bits uncoded can affect performance.

This figure shows a general trellis coder. The incoming source bits are divided into $\hat{m}$ bits to be coded and $m - \hat{m}$ bits to be left uncoded. The required convolutional coder has rate $\hat{m} / (\hat{m} + 1)$, and the trellis will have $2^{m - \hat{m}}$ parallel transitions between every pair of states.
Bit Mapping for Trellis Codes

To complete the coder design, we need to design the mapping of bits into symbols. Designing the mapping is the same as assigning symbols to transitions in the trellis diagram.

When there are parallel transitions, a very short error event consist of mistaking one of these parallel transitions for the correct one. To minimize error probability, we should ensure that the Euclidean distance between the symbols corresponding to parallel transitions is maximized.

For example with 8-PSK, this can done as illustrated in the following.

The constellation is divided into four subsets. To ensure that parallel transitions in trellis have symbols as far apart as possible, symbols for parallel transitions are selected form the same subset. Now, assign subsets to pairs of transitions to try to maximize the minimum distance. This can be done by keeping the distance between diverging or merging branches as large as possible.

The performance of the code is dominated by the error event with the smallest distance from the correct path.
Using mapping for 8-PSK explained above, the distance between parallel transitions is $2a$ ($a$ is the amplitude of 8-PSK), and this is the minimum distance. The probability of error event is given as:

$$\Pr(\text{error event}) = Q\left(\frac{a}{\sigma}\right)$$

The error events result in exactly one bit error out of two uncoded bits, and there is only one such event for each correct path, thus:

$$\Pr(\text{bit error}) \approx \frac{1}{2} Q\left(\frac{a}{\sigma}\right)$$

This can be compared to the uncoded system, which corresponds to 4-PSK:

$$\Pr(\text{bit error}) = Q\left(\frac{\sqrt{2}a}{\sigma}\right)$$

Ignoring the constant multiplier, the coding gain is: $20 \log \sqrt{2} = 3 dB$, and this is 1dB lower compared to the system without parallel transitions.
Mapping by Set Partitioning

A systematic way to design bit mappings in such way that the parallel transitions correspond to symbols that are as far apart as possible is generally known as mapping by set partitioning.

Once the alphabet is chosen and the number of coded and uncoded bits selected, we still have freedom to map the coded bits into symbols. The choice of mapping can affect the performance of the code.

**Example 1:** Systematic partitioning of an 8-PSK constellation

![8-PSK constellation diagram]

**Example 2:** Systematic partitioning of an 16-QAM constellation

![16-QAM constellation diagram]

Note that the uncoded bits select a signal from subset, and the coded bits select the subset.

The general rules for mapping by set partitioning are:
- maximize the distance between parallel transitions first
- maximize the distance between transitions originating or ending in the same state
- use all symbols with equal probability.
Mapping by Set Partitioning: Example

A trellis coder with one uncoded bit and two coded bits with 16-QAM from previous slide. The convolutional coder is an 8 state, rate 2/3, and of feedback type.

The corresponding trellis diagram:

The coding gain of this coder is about 5.33 dB. If one more uncoded bit is used with 32-QAM, the coding gain reduces to 3.98 dB, with two more uncoded bits (64-QAM) the coding gain reduces to 3.77 dB.
Catastrophic Codes

In most cases we have assumed that there is only one minimum distance error event, and it dominates the performance. The code that has unbounded number of minimum distance error events is called catastrophic.

Example: A four point constellation with partition shown below. We consider transmitting 2 bits per symbol. This can be represented by using two state trellis. We can see that there is infinite number of error events with same (finite) minimum distance. In this case, we should not expect any coding gain at all, because we are representing two source bits with an alphabet of size four (no redundancy).

Catastrophic codes have property that there are error events with finite distance but infinite length.
Multidimensional Trellis Codes

Some of the most interesting trellis codes in use are multidimensional trellis codes. The idea of these codes is to overcome one of the main limitations of trellis codes, the requirement that the size of the symbol alphabet be doubled.

(Recall the observation that doubling the symbol alphabet is sufficient to achieve almost all available coding gain). However, doubling the alphabet for the same average power reduces the spacing between symbols and reduces noise margin by about 3 dB.

In case of a multidimensional trellis code, doubling the alphabet entails less than 3 dB loss in noise margin.

Example: We consider a four dimensional symbol

\[ A_k = [a_1, a_2, a_3, a_4], \text{ where } a_i \in \{\pm 3, \pm 1\} \]

There are 256 symbols in this alphabet, with average power 20. A four dimensional alphabet with 512 symbols and average power 27 is constructed by adding symbols of the form \( \{\pm 5, \pm 3, \pm 1, \pm 1\} \), and all its permutations, and symbols of form \( \{\pm 5, \pm 1, \pm 1, \pm 1\} \) and all its permutations.

Given a two dimensional modulation system the first two coefficients of the four dimensional system are transmitted in one symbol interval as the real and imaginary part of a complex symbol, and the last two in the next symbol interval.

The trellis coder that makes use of this is shown here. The four dimensional alphabet has 512 symbols, and can represent 9 bits. Three bits are coded to get four bits, and five bits are used uncoded. The coding gain of this structure is about 4.7dB. The two dimensional trellis coder with same source bit rate has coding gain of 4dB.
Trellis Codes using Non-Linear Convolutional Codes

Some desirable properties can be obtained by using non-linear convolutional codes.

One example coder is the following, which is adopted in the ITU-T recommendation V.32 for voiceband modem. The main advantage of this code is that it is invariant over 90 degree phase shifts, which is a big advantage in the context of carrier recovery. Some of symmetry is evident from constellation. It is evident that \( C_i^{(1)} \) and \( C_i^{(2)} \) are the same at all points 90 degrees apart. Furthermore, \( C_i^{(3)} \) and \( C_i^{(4)} \) differentially encode the quadrant, i.e., to move for -90 degrees, simply add one to these two bits. This differential encoding is introduced by the differential encoder in the figure.