Notes on Coding

1. Introduction

It is known that at reasonably high signal to noise ratios (SNRs), probability of error is primarily determined by the minimum distance between signal vector ends. The philosophy of coding is to (artificially) increase this minimum distance. One way of doing this is to increase the dimensionality of the signal space and leave some signal vector positions unoccupied, so the minimum distance between the adjacent signal vector ends is physically increased, thus reducing probability of error.

We illustrate this point by Example 9.4.1 of Proakis 2002.

Example 1.1 : Assume that we have 4 PSK whose constellation diagram is as shown in Fig. 1.1 together with (common) length of signal vectors the respective distances between them.

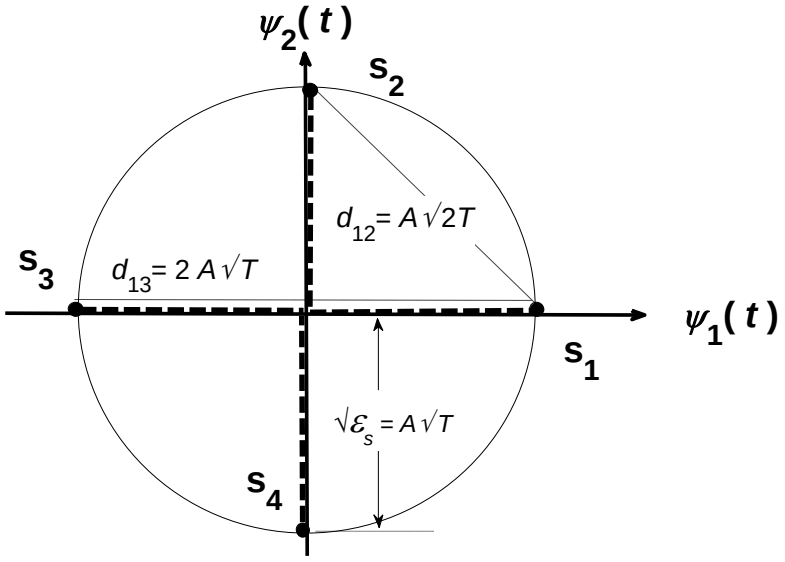


Fig. 1.1 Constellation diagram for 4 PSK (two dimensional case).

For clarity, we state below the properties of 4 PSK shown in Fig. 1

$$d_{2\min 4\text{PSK}} = d_{12} = d_{14} = d_{23} = A\sqrt{2T}\ ,\ d_{13} = d_{24} = 2A\sqrt{T}$$
$$|\mathbf{s}_1| = |\mathbf{s}_2| = |\mathbf{s}_3| = |\mathbf{s}_4| = A\sqrt{T}\ \ ,\ \ \varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon_4 = \varepsilon_s = A^2T \qquad (1)$$

As pointed out above, the major contributing part to probability of error for the constellation of Fig. 1.1 will come from the cases of, say $\mathbf{s}_1$ being transmitted and this being wrongly detected as $\mathbf{s}_2$ or $\mathbf{s}_4$ (not $\mathbf{s}_3$) on the receiver side. Thus it is imperative that we concentrate on increasing $d_{2\min 4\text{PSK}} = d_{12} = d_{14} = d_{23}$.

Suppose now that we wish to design another constellation diagram which has a larger minimum distance between the adjacent signal vector ends than $d_{2\min 4\text{PSK}}$ given in (1.1) (Keeping the signal

vector lengths or the total or the average energy the same). One way to do this is to increase the dimensionality of the signal space from two to three. Such a configuration is shown in Fig. 1.2.

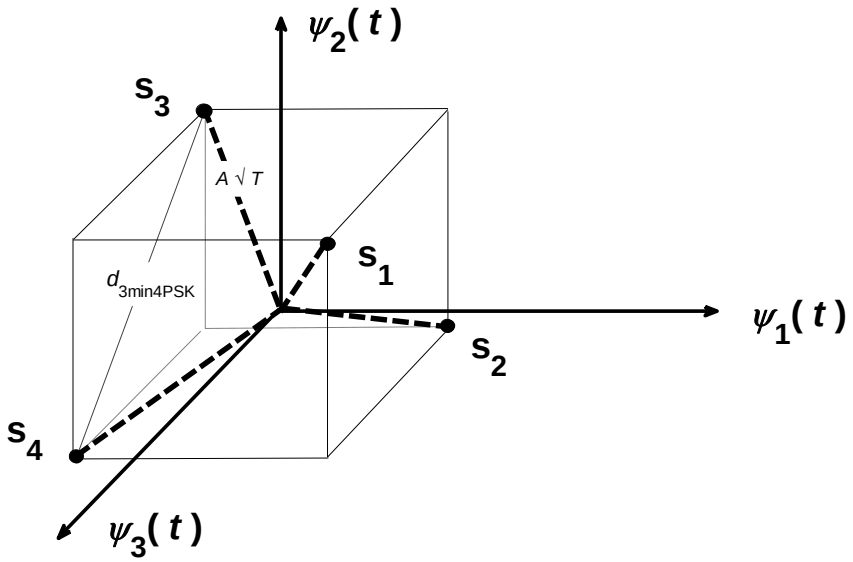


Fig. 1.2 Constellation diagram for 4 PSK (three dimensional case).

In Fig. 1.2, we have placed the signal vectors on the vertices (corners) of a cube whose centre coincides with the origin of the three dimensional signal space. Additionally, it is arranged that the distances between the signal vector ends are maximized by placing our signals diagonally on cube sides and leaving the other diagonals unoccupied. To see the amount of increased separation we have gained between signal vector ends by going from two to three dimensions, we write for the signal vector components of the three dimensional case as follows

$$\begin{aligned}
\mathbf{s}_1 &= \left[+\sqrt{\varepsilon_c},\ +\sqrt{\varepsilon_c},\ +\sqrt{\varepsilon_c}\right] = \sqrt{\varepsilon_c}\left[+1,\ +1,\ +1\right] \\
\mathbf{s}_2 &= \left[+\sqrt{\varepsilon_c},\ -\sqrt{\varepsilon_c},\ -\sqrt{\varepsilon_c}\right] = \sqrt{\varepsilon_c}\left[+1,\ -1,\ -1\right] \\
\mathbf{s}_3 &= \left[-\sqrt{\varepsilon_c},\ -\sqrt{\varepsilon_c},\ +\sqrt{\varepsilon_c}\right] = \sqrt{\varepsilon_c}\left[-1,\ -1,\ +1\right] \\
\mathbf{s}_4 &= \left[-\sqrt{\varepsilon_c},\ +\sqrt{\varepsilon_c},\ -\sqrt{\varepsilon_c}\right] = \sqrt{\varepsilon_c}\left[-1,\ +1,\ -1\right]
\end{aligned} \tag{1.2}$$

It is easy to find what ε_c should be in terms of $\varepsilon_s = A^2T$ so that there is energy equivalence between the constellation of Figs. 1.1 and 1.2. Hence

$$\varepsilon_1 = \|\mathbf{s}_1\|^2 = 3\varepsilon_c = A^2T,\ \ \varepsilon_c = \frac{A^2T}{3} \tag{1.3}$$

This way, we can find $d_{3\min 4\text{PSK}}$ of the constellation in Fig. 1.3 and compare it to $d_{2\min 4\text{PSK}}$ of constellation of Fig. 1.2 as

$$d^2_{3\min 4\text{PSK}} = 8\varepsilon_c = \frac{8\varepsilon_s}{3} = \frac{8A^2T}{3}$$

$$d_{3\min 4\text{PSK}} = 2A\sqrt{\frac{2T}{3}} = 1.633A\sqrt{T},\ d_{2\min 4\text{PSK}} = A\sqrt{2T} = 1.4142A\sqrt{T}$$

$$\frac{d_{3\min 4\text{PSK}}}{d_{2\min 4\text{PSK}}} = 1.1547 \tag{1.4}$$

As seen from (1.4), we have an increase by a factor of 1.1547 in the minimum distance, upon going from two dimensions to three dimensions. If we were to fill the unoccupied signal positions in Fig. 1.2, we would obtain something similar to 8 PSK, but in three dimensions. Using Fig. 3.5 of the Notes on Dimensionality of Signals_Sept 2012 document and Fig. 1.2 of this document, it is possible to estimate the minimum distances in the 8 PSK cases of two and three dimensions as follows (in both cases, we keep the energies of the signal vectors fixed at $\varepsilon_s = A^2T$)

$$d_{3\min 8\text{PSK}} = 2A\sqrt{\frac{T}{3}} = 1.1547A\sqrt{T},\ d_{2\min 8\text{PSK}} = A\sqrt{T\left(2-\sqrt{2}\right)} = 0.7654A\sqrt{T}$$

$$\frac{d_{3\min 8\text{PSK}}}{d_{2\min 8\text{PSK}}} = 1.5087 \tag{1.5}$$

Thus increasing the dimensionality always increases the minimum distance, but we should keep in mind that with increasing number of dimensions, our bandwidth requirements and receiver complexity rise as well.

Exercise 1.1 : By using Fig. 1.2 of this document and Fig. 3.5 of the Notes on Dimensionality of Signals_Sept 2012 document (which can be reached from Lecture Notes of ECE 632 course webpage), verify $d_{3\min 8\text{PSK}}$ and $d_{2\min 8\text{PSK}}$ in given in (1.5).

Exercise 1.2 : Find the minimum distance between vector ends in Fig. 1.2, if we were to use eight signal vectors (instead of four), thus fill in all vertices (corners) of the cube.

It is interesting to note that, in the present arrangement, four vertices of the cube in Fig. 1.2 remain unoccupied. This is exactly what coding does, that is, coding increases the dimensionality of the signal space, leaving the number of symbols in our alphabet the same, i.e. Figs. 1.1 and 1.2 both refer to the case of 4 PSK. Another interpretation is that Fig. 1.2 represents a signal space of eight symbols, but only four symbols ("codewords" in the terminology of coding) out of the eight are valid symbols, the remaining four symbols are invalid symbols of the message signal. We describe coding in terms of binary waveforms (bits) rather than symbols. Hence in the present example, when going from 4 PSK to 8 symbols configuration, we have extended the existing two bit length of the message signal to three bits. In general we represent the length of the uncoded message signal as k bits, while the coded message will have a length of n bits, where it is always the case that $n > k$.

Normally coding is studied in terms of binary bits. And the symbols are named as code words. In line with this terminology, we contemplate that the (two dimensional) 4 PSK of Fig. 1.1 represents the situation prior to coding, while (three dimensional) 4 PSK of Fig. 1.2 is the coded case. In 4 PSK, $M = 4$ (number of signals in the set). Thus it is natural to assume that we have

grouped the binary waveform two by two. Let the assignment of this grouping to signal vectors $\mathbf{s}_1,\ \mathbf{s}_2,\ \mathbf{s}_3,\ \mathbf{s}_4$ be

$$\begin{aligned}\mathbf{s}_1 &= 00\\ \mathbf{s}_2 &= 01\\ \mathbf{s}_3 &= 10\\ \mathbf{s}_4 &= 11\end{aligned} \qquad (1.6)$$

Now the constellation in Fig. 1.2 is still 4 PSK, but as mentioned above, it has 8 signalling positions that could be used. Since we have sought an increase in the minimum distance between signal vector ends, we have only occupied only four of them. This means we have gone from $k=2$ (uncoded) bits to $n=3$(coded) bits. Below we show in the first column, a possible assignment of $n=3$bit symbols (code words) to four signal vectors and in the second column, all possible symbols that could be created by grouping three binary bits.

Assignment of three bits to four sysmbols	All possible symbols of three bits
$\mathbf{s}_1 = 000$	000
$\mathbf{s}_2 = 011$	001
$\mathbf{s}_3 = 101$	010
$\mathbf{s}_4 = 110$	011
	100
	101
	110
	111

(1.7)

By comparing the first column of (1.7) with (1.6), we see that, while the symbols (signal vectors) of (1.6) differ from each other in one bit, in (1.7), this difference has gone up to two bits (of the same time location). This is another indication of increasing the minimum distance when we go from two dimensional 4 PSK to three dimensional 4 PSK. Another observation from (1.7) is that, symbols such as $001, 010, 100, 110$ are not among the valid symbols of the three dimensional PSK. From this point onwards, we drop the reference to PSK, and describe coding in its own terminology.

Accepting that symbols (consisting of two bits) in (1.6) are the uncoded message signals and the symbols (consisting of three bits) belonging to the first column of (1.7) represent the coded coded symbols, then we see that there must be a mapping mechanism that will somehow relate the uncoded message symbols to coded ones. Such mechanisms may be rather simplified as in the case of linear and cyclic codes. They may also be complex ones, involving dependence on the present as well past symbols, thus possessing memory. One such scheme is convolutional coding. Next we study these codes in turn.

2. Linear Block Codes

2.1 Basic Definitions

We adopt the following notations for a code

$$\begin{aligned}&\text{Uncoded message} \quad : X = \{\mathbf{x}_1,\ \mathbf{x}_2,\ \cdots\ \mathbf{x}_i,\ \cdots\ \mathbf{x}_M\}\\ &\text{Coded message (code)} : \ C = \{\mathbf{c}_1,\ \mathbf{c}_2,\ \cdots\ \ \mathbf{c}_i,\ \cdots\ \mathbf{c}_M\}\end{aligned} \tag{2.1}$$

where X is the complete collection of the (uncoded) symbols in the message signal set, C is the coded version of X. The number of code words in C and the number of symbols in the uncoded message are both M. $\mathbf{x}_i$ is one of the information sequences in the uncoded message X, while $\mathbf{c}_i$ is one of the code words in the code C. For any $\mathbf{x}_i$ and $\mathbf{c}_i$ with $1 \le i \le 2^k = M$, we have

$$\begin{aligned}\mathbf{x}_i &= \left({}_i x_1,\ {}_i x_2,\ \cdots,\ {}_i x_j,\ \cdots,\ {}_i x_k\right)\\ \mathbf{c}_i &= \left({}_i c_1,\ {}_i c_2,\ \cdots\cdots,\ {}_i c_j,\ \cdots\cdots,\ {}_i c_n\right) \quad , \quad n > k\end{aligned} \tag{2.2}$$

so x_j and c_j are the individual components (bits) of $\mathbf{x}_i$ and $\mathbf{c}_i$ which are simply 0 or 1.

A block code is linear if any linear combinations of code words is another code word in the same code. So let $\mathbf{c}_i,\ \mathbf{c}_j,\ \mathbf{c}_k$ be code any words selected from the code given in (2.1), then if

$$\mathbf{c}_i \oplus \mathbf{c}_j = \mathbf{c}_k \quad , \quad 0 \le i \le M \quad , \quad 0 \le j \le M \quad , \quad 0 \le k \le M \tag{2.3}$$

the code C is said to be linear. In (2.3) $\oplus$ indicates componentwise modulo-2 addition operating according to the following simple binary addition rules.

$$0 \oplus 0 = 0 \quad , \quad 1 \oplus 0 = 0 \oplus 1 = 1 \quad , \quad 1 \oplus 1 = 0 \tag{2.4}$$

From (2.1), we may assume that if the information sequence $\mathbf{x}_1$ is mapped to $\mathbf{c}_1$ and if the information sequence $\mathbf{x}_2$ is mapped to $\mathbf{c}_2$, then it is natural to expect that $\mathbf{x}_1 \oplus \mathbf{x}_2$ is mapped to $\mathbf{c}_1 \oplus \mathbf{c}_2$. This is a special property which may or may not be present in a linear block code.

Example 2.1 : A $(n,\ k) = (5,\ 2)$ code is defined as

$$\begin{aligned}&\quad \mathbf{c}_1 \quad\ \ \mathbf{c}_2 \quad\ \ \mathbf{c}_3 \quad\ \ \mathbf{c}_4\\ &C = \{00000,\ 10100,\ 01111,\ 11011\}\end{aligned} \tag{2.5}$$

It is easy to verify that the code given in (2.5) is linear by carrying out the following computations,

$$\begin{aligned}&\mathbf{c}_1 \oplus \mathbf{c}_1 = \mathbf{c}_1 \quad , \quad \mathbf{c}_2 \oplus \mathbf{c}_2 = \mathbf{c}_1 \quad , \quad \mathbf{c}_3 \oplus \mathbf{c}_3 = \mathbf{c}_1 \quad , \quad \mathbf{c}_4 \oplus \mathbf{c}_4 = \mathbf{c}_1\\ &\mathbf{c}_1 \oplus \mathbf{c}_2 = \mathbf{c}_2 \quad , \quad \mathbf{c}_1 \oplus \mathbf{c}_3 = \mathbf{c}_3 \quad , \quad \mathbf{c}_1 \oplus \mathbf{c}_4 = \mathbf{c}_4\\ &\mathbf{c}_2 \oplus \mathbf{c}_3 = \mathbf{c}_4 \quad , \quad \mathbf{c}_2 \oplus \mathbf{c}_4 = \mathbf{c}_3 \quad , \quad \mathbf{c}_3 \oplus \mathbf{c}_4 = \mathbf{c}_2\\ &\text{Sample computation : } \mathbf{c}_2 \oplus \mathbf{c}_3 = 1\ 0\ 1\ 0\ 0 \quad \leftarrow \text{Componentwise}\\ &\qquad\qquad\qquad\qquad\qquad\quad \underline{0\ 1\ 1\ 1\ 1} \quad \leftarrow \text{modulo-2 addition}\\ &\qquad\qquad\qquad\qquad\qquad\quad 1\ 1\ 0\ 1\ 1 \ = \mathbf{c}_4\end{aligned} \tag{2.6}$$

To find whether the code given in (2.5) satisfies the special property described underneath (2.4), we have to know the mapping between X and C. In this context, if this mapping is

$$\begin{array}{lcl} X & & C \\ \mathbf{x}_1 = 00 & \rightarrow & 00000 = \mathbf{c}_1 \\ \mathbf{x}_2 = 01 & \rightarrow & 01111 = \mathbf{c}_3 \\ \mathbf{x}_3 = 10 & \rightarrow & 10100 = \mathbf{c}_2 \\ \mathbf{x}_4 = 11 & \rightarrow & 11011 = \mathbf{c}_4 \end{array} \tag{2.7}$$

then, the special property of $\mathbf{x}_i$ being mapped to $\mathbf{c}_i$ and $\mathbf{x}_j$ being mapped to $\mathbf{c}_j$, and $\mathbf{x}_i \oplus \mathbf{x}_j$ being mapped to $\mathbf{c}_i \oplus \mathbf{c}_j$ is satisfied. But the following mapping for instance does not satisfy this criteria.

$$\begin{array}{lcl} X & & C \\ \mathbf{x}_1 = 00 & \rightarrow & 10100 = \mathbf{c}_2 \\ \mathbf{x}_2 = 01 & \rightarrow & 01111 = \mathbf{c}_3 \\ \mathbf{x}_3 = 10 & \rightarrow & 00000 = \mathbf{c}_1 \\ \mathbf{x}_4 = 11 & \rightarrow & 11011 = \mathbf{c}_4 \end{array} \tag{2.8}$$

Exercise 2.1 : Verify by hand that the mapping in (2.7) does indeed satisfy and the mapping in (2.8) does not satisfy the special property for all information sequences and code words of Example (2.1).

Some important measures of a code are listed below.

Hamming distance : This is the number of components that differ between any two code words, $\mathbf{c}_i$ and $\mathbf{c}_j$. For instance the Hamming distance between $\mathbf{c}_1$ and $\mathbf{c}_2$ of Example 2.1 is

$$d(\mathbf{c}_1, \mathbf{c}_2) = 2 \tag{2.9}$$

Hamming weight : This is the number of nonzero components in the code word and denoted by $w(\mathbf{c}_i)$. For instance the Hamming weight of $\mathbf{c}_2$ of Example 2.1 is

$$w(\mathbf{c}_2) = 2 \tag{2.10}$$

We can also define the minimum weight, which is the minimum of the code word weight except the all zero code word, hence

$$w_{\min} = \min_{c_i \neq 0} \left[w(\mathbf{c}_i) \right] \tag{2.11}$$

Minimum distance : This is the minimum Hamming distance between two different code words, $\mathbf{c}_i$ and $\mathbf{c}_j$. Thus in terms of (2.9), this would be

$$d_{\min} = \min_{\substack{\mathbf{c}_i, \mathbf{c}_j \\ i \neq j}} \left[d\left(\mathbf{c}_i, \mathbf{c}_j\right) \right] \tag{2.12}$$

Exercise 2.2 : For the $(5,\ 2)$ code given in Example 2.1, find the Hamming distance, weights of all code words. Also evaluate the minimum distance and the minimum weight, proving that $w_{\min} = d_{\min}$.

2.2 Generator Matrix for Linear Block Codes

By identifying the code words generated by the information sequences, $\mathbf{x}_{g1} = (1000\cdots0)$, $\mathbf{x}_{g2} = (0100\cdots0)$, $\mathbf{x}_{g3} = (0010\cdots0)$ $\cdots$ $\mathbf{x}_{gk} = (0000\cdots1)$ as $\mathbf{c}_{g1}$, $\mathbf{c}_{g2}$, c_{g3}, $\cdots$, $\mathbf{c}_{gk}$ it is possible to for a generator matrix $\mathbf{G}$ such that

$$\mathbf{G} = \begin{bmatrix} \mathbf{c}_{g1} \\ \mathbf{c}_{g2} \\ \mathrm{c}_{g3} \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{c}_{gk} \end{bmatrix} = \begin{bmatrix} c_{g11} & c_{g12} & c_{g13} & \cdots & c_{g1n} \\ c_{g21} & c_{g22} & c_{g23} & \cdots & c_{g2n} \\ c_{g31} & c_{g32} & c_{g33} & \cdots & c_{g3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_{gk1} & c_{gk2} & c_{gk3} & \cdots & c_{gkn} \end{bmatrix} \tag{2.13}$$

Eventually, the code word or the code can be obtained from the multiplication of the message by the generator polynomial, that is

$$\begin{aligned} \mathbf{c} &= \mathbf{x}\mathbf{G} \\ C &= X\mathbf{G} \end{aligned} \tag{2.14}$$

Example 2.2 : To find the generator matrix, $\mathbf{G}$ of the $(5,\ 2)$ code given in Example 2.1 (the one that has the special property), we see that $\mathbf{x}_{g1} = \mathbf{x}_3 = (10)$, $\mathbf{x}_{g2} = \mathbf{x}_2 = (01)$, thus, the corresponding code words are, $\mathbf{c}_{g1} = \mathbf{c}_2 = (10100)$, $\mathbf{c}_{g2} = \mathbf{c}_3 = (01111)$. Therefore the generator matrix becomes

$$\mathbf{G} = \begin{bmatrix} \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix} = \begin{bmatrix} 1\ 0\ 1\ 0\ 0 \\ 0\ 1\ 1\ 1\ 1 \end{bmatrix} \tag{2.15}$$

For a single codeword, (2.14) is in the form of

$$(c_1,\ c_2,\ c_3,\ c_4,\ c_5) = (x_1,\ x_2)\mathbf{G} \tag{2.16}$$

Considering the specific numeric values of $\mathbf{G}$ matrix, (2.16) will turn into

$$
\begin{aligned}
c_1 &= x_1 \\
c_2 &= x_2 \\
c_3 &= x_1 \oplus x_2 \\
c_4 &= x_2 \\
c_5 &= x_2
\end{aligned} \tag{2.17}
$$

From (2.17), we understand that the code word consists of two distinct parts, the first part, that is c_1 and c_2 are obtained directly from the first two bits of the information sequence, i.e., x_1 and x_2, the remaining three bits, that is, c_3, c_4, c_5, also called parity check bits, are linear combinations of x_1 and x_2. Such a code is called a systematic code and its generator matrix is expressed in the form

$$
\mathbf{G} = \left[\mathbf{I}_k \middle| \mathbf{P}\right] = \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{array}\right] \quad , \quad \mathbf{I}_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad , \quad \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.18}
$$

From (2.18), we see that the first part of $\mathbf{G}$, i.e., $\mathbf{I}_k$ is a $k \times k$ identity matrix, and the second part, i.e., $\mathbf{P}$ is $k \times (n-k)$ matrix, determining how (the added) parity bits will depend on the bits of the information sequence.

The dual code of C, is denoted by C^T. Let the generator matrix of C^T be $\mathbf{H}$, also called the parity check matrix, then it is possible to retrieve $\mathbf{H}$ from the two parts of $\mathbf{G}$ in the following manner

$$
\mathbf{H} = \left[\mathbf{P}^t \middle| \mathbf{I}_{n-k}\right] \tag{2.19}
$$

where $\mathbf{P}^t$ is the transpose of $\mathbf{P}$. For parity check matrix $\mathbf{H}$, the following relations will hold

$$
\mathbf{G}\mathbf{H}^t = 0 \quad , \quad \mathbf{c}\mathbf{H}^t = 0 \tag{2.20}
$$

Example 2.3 : For the code given in Example 2.1, find the parity check matrix $\mathbf{H}$.

Solution : From (2.18), we get

$$
\mathbf{I}_k = \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad , \quad \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad , \quad \mathbf{I}_{n-k} = \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad \mathbf{P}^t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}
$$

$$
\mathbf{H} = \left[\mathbf{P}^t \middle| \mathbf{I}_{n-k}\right] = \left[\mathbf{P}^t \middle| \mathbf{I}_3\right] = \left[\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{2.21}
$$

Exercise 2. 3 : By taking the code of Example 2.1, prove the identities in (2.20). This can be done by hand or using the Matlab code LBcode_Sub2013.m available on course webpage.

Exercise 2. 4 : Either by hand or by running the Matlab code LBcode_Sub2013.m available on course webpage, construct a $(5,\ 3)$ linear block code. For this purpose use $\mathbf{P}$ given in Matlab code LBcode_Sub2013.m. Test the linear block code property of this code by carrying out computations as in (2.6). Find the Hamming distance, weights of all code words. Also evaluate the minimum distance and the minimum weight, proving that $w_{\min} = d_{\min}$. Prove the identities in (2.20). Investigate other $(5,\ 3)$ linear block codes by changing $\mathbf{P}$ and make new evaluations of Hamming distance and weights of all code words. Comment on the changes you observe with this new $\mathbf{P}$, in these quantities.

3. Cyclic Codes

3.1 Basic Definitions

A cyclic code is a linear block code with the extra property that for any code word $\mathbf{c}_i$, its cyclic shift is also a code word in the same code. For a code word $\mathbf{c}_i$, its j th cyclic shift is defined as

$$\begin{aligned}
&\mathbf{c}_i = \left({}_ic_1,\ {}_ic_2,\ \cdots\cdots,\ {}_ic_j,\ \cdots\cdots,\ {}_ic_n\right) \\
&\mathbf{c}_i^{(j)} = \left({}_ic_{j+1},\ {}_ic_{j+2},\ \cdots\cdots,\ {}_ic_n,\ {}_ic_1,\ {}_ic_2,\ \cdots\cdots,\ {}_ic_j\right) \qquad , \qquad 0 \geq j \geq n-1
\end{aligned} \tag{3.1}$$

Example 3.1 : It can be verified that the following code is linear and cyclic

$$\begin{aligned}
&{}_1\mathbf{c}_1 \quad {}_1\mathbf{c}_2 \quad {}_1\mathbf{c}_3 \quad {}_1\mathbf{c}_4 \qquad {}_1\mathbf{c}_1^{(0)} \quad {}_1\mathbf{c}_2^{(0)} \quad {}_1\mathbf{c}_3^{(0)} \quad {}_1\mathbf{c}_4^{(0)} \\
&C_1 = \{000,\ 110,\ 101,\ 011\} = \{000,\ 110,\ 101,\ 011\}
\end{aligned} \tag{3.2}$$

The cyclic property for the code in (3.2) can be proven as listed below

$$\begin{aligned}
&{}_1\mathbf{c}_1^{(0)} = {}_1\mathbf{c}_1^{(1)} = {}_1\mathbf{c}_1^{(2)} = {}_1\mathbf{c}_1^{(3)} = (000) \\
&{}_1\mathbf{c}_2^{(0)} = (110) \ , \ {}_1\mathbf{c}_2^{(1)} = (101) = {}_1\mathbf{c}_3^{(0)} \ , \ {}_1\mathbf{c}_2^{(2)} = (011) = {}_1\mathbf{c}_4^{(0)} \\
&{}_1\mathbf{c}_3^{(0)} = (101) \ , \ {}_1\mathbf{c}_3^{(1)} = (011) = {}_1\mathbf{c}_4^{(0)} \ , \ {}_1\mathbf{c}_3^{(2)} = (101) = {}_1\mathbf{c}_3^{(0)} \\
&{}_1\mathbf{c}_4^{(0)} = (011) \ , \ {}_1\mathbf{c}_4^{(1)} = (110) = {}_1\mathbf{c}_2^{(0)} \ , \ {}_1\mathbf{c}_4^{(2)} = (101) = {}_1\mathbf{c}_3^{(0)}
\end{aligned} \tag{3.3}$$

The next code, C_2 is linear, but not cyclic as illustrated below

$$\begin{aligned}
&{}_2\mathbf{c}_1 \quad {}_2\mathbf{c}_2 \quad {}_2\mathbf{c}_3 \quad {}_2\mathbf{c}_4 \qquad {}_2\mathbf{c}_1^{(0)} \quad {}_2\mathbf{c}_2^{(0)} \quad {}_2\mathbf{c}_3^{(0)} \quad {}_2\mathbf{c}_4^{(0)} \\
&C_2 = \{000,\ 010,\ 101,\ 111\} = \{000,\ 010,\ 101,\ 111\} \\
&{}_2\mathbf{c}_1^{(0)} = {}_2\mathbf{c}_1^{(1)} = {}_2\mathbf{c}_1^{(2)} = {}_2\mathbf{c}_1^{(3)} = (000) \\
&{}_2\mathbf{c}_2^{(0)} = (010) \ , \ {}_2\mathbf{c}_2^{(1)} = (100) = \text{NE} \ , \ {}_2\mathbf{c}_2^{(2)} = (001) = \text{NE} \\
&{}_2\mathbf{c}_3^{(0)} = (101) \ , \ {}_2\mathbf{c}_3^{(1)} = (011) = \text{NE} \ , \ {}_2\mathbf{c}_3^{(2)} = (110) = \text{NE} \\
&{}_2\mathbf{c}_4^{(0)} = {}_2\mathbf{c}_4^{(1)} = {}_2\mathbf{c}_4^{(2)} = {}_2\mathbf{c}_4^{(3)} = (111) \quad , \quad \text{NE} = \text{ No Equivalence}
\end{aligned} \tag{3.4}$$

Exercise 3.1 : Prove that codes, C_1 and C_2 given in (3.2) and (3.4) are linear block code.

3.2 Generator Polynomial of Cyclic Codes

For a $(n,\ k)$ cyclic code, the generator polynomial, is of the degree of $n-k$ and has the form

$$g(p)=g_1p^{n-k}+g_2p^{n-k-1}+g_3p^{n-k-2}+\ \cdots\ +g_{n-k}p+1 \tag{3.5}$$

and it divides in modulo-2 sense the polynomial p^n+1 without remainder. This means when p^n+1 is factorized, $g(p)$ is one of the multiplicative polynomials in this factorization. To find a particular code word $\mathbf{c}=(c_1,\ c_2,\ \cdots\cdots,\ c_j,\ \cdots\cdots,\ c_n)$ corresponding to the information sequence, $\mathbf{x}=(x_1,\ x_2,\ \cdots,\ x_j,\ \cdots,\ x_k)$, we initially express the information sequence in the following polynomial form

$$X(p)=x_1p^{k-1}+x_2p^{k-2}+x_3p^{k-3}+\ \cdots\ +x_{k-1}p+x_k \tag{3.6}$$

Then the code word $\mathbf{c}$ is written from the product of the code word polynomial $c(p)$ and the generator polynomial $g(p)$, thus

$$\mathbf{c}=\text{coefficients of } p \text{ in } c(p)\ ,\ \text{ where } c(p)=X(p)g(p) \tag{3.7}$$

Example 3.2 : To generate $(n=7,\ k=4)$ cyclic code, we take the polynomial, p^7+1 and factorize it as follows

$$p^7+1=(p+1)(p^3+p^2+1)(p^3+p+1)$$

$$\begin{array}{l|l}
p^7+1 & \qquad\qquad\qquad\qquad\qquad\qquad p^3+p^2+1 \\
\underline{p^7+p^6+p^4} & \qquad\qquad\qquad\qquad\qquad\qquad \overline{)\ p^4+p^3+p^2+1=(p+1)(p^3+p+1)} \\
0+p^6+p^4+1 & \\
\quad\ \underline{p^6+p^5+p^3} & \\
p^5+p^4+p^3+1 & \\
\underline{p^5+p^4+p^2} & \\
\quad\ p^3+p^2+1 & \\
\quad\ \underline{p^3+p^2+1} & \\
\quad\ 0\quad 0\quad 0\ \leftarrow \text{ remainder} &
\end{array} \tag{3.8}$$

As shown in (3.8), there are two polynomials of the degree $n-k=3$, namely, p^3+p^2+1 and p^3+p+1, by choosing $g(p)=p^3+p^2+1$, from (3.6) we get

$$X(p)=x_1p^3+x_2p^2+x_3p+x_4 \tag{3.9}$$

Below, in Table 3.1, we list the complete symbols of the message signal, and the corresponding code words generated using generator polynomial, $g(p) = p^3 + p^2 + 1$.

Message signal symbols, information sequences $X = \{\mathbf{x}_1,\ \mathbf{x}_2,\ \cdots\ \mathbf{x}_i,\ \cdots\ \mathbf{x}_{16}\}$						$X(p)$	$c(p) = X(p)g(p)$	Code words of code $C = \{\mathbf{c}_1,\ \mathbf{c}_2,\ \cdots\ \mathbf{c}_i,\ \cdots\ \mathbf{c}_{16}\}$							
i	$\mathbf{x}_i$	${}_i x_1$	${}_i x_2$	${}_i x_3$	${}_i x_4$			$\mathbf{c}_i$	${}_i c_1$ p^6	${}_i c_2$ p^5	${}_i c_3$ p^4	${}_i c_4$ p^3	${}_i c_5$ p^2	${}_i c_6$ p^1	${}_i c_7$ p^0
1	0000	0	0	0	0	0	0	0000000	0	0	0	0	0	0	0
2	0001	0	0	0	1	1	$p^3 + p^2 + 1$	0001101	0	0	0	1	1	0	1
3	0010	0	0	1	0	p	$p^4 + p^3 + p$	0011010	0	0	1	1	0	1	0
4	0100	0	1	0	0	p^2	$p^5 + p^4 + p^2$	0110100	0	1	1	0	1	0	0
5	1000	1	0	0	0	p^3	$p^6 + p^5 + p^3$	1101000	1	1	0	1	0	0	0
6	0011	0	0	1	1	$p+1$	$p^4 + p^2 + p + 1$	0010111	0	0	1	0	1	1	1
7	0110	0	1	1	0	$p^2 + p$	$p^5 + p^3 + p^2 + p$	0101110	0	1	0	1	1	1	0
8	1100	1	1	0	0	$p^3 + p^2$	$p^6 + p^4 + p^3 + p^2$	1011100	1	0	1	1	1	0	0
9	1001	1	0	0	1	$p^3 + 1$	$p^6 + p^5 + p^2 + 1$	1100101	1	1	0	0	1	0	1
10	0101	0	1	0	1	$p^2 + 1$	$p^5 + p^4 + p^3 + 1$	0111001	0	1	1	1	0	0	1
11	1010	1	0	1	0	$p^3 + p$	$p^6 + p^5 + p^4 + p$	1110010	1	1	1	0	0	1	0
12	0111	0	1	1	1	$p^2 + p + 1$	$p^5 + p + 1$	0100011	0	1	0	0	0	1	1
13	1110	1	1	1	0	$p^3 + p^2 + p$	$p^6 + p^2 + p$	1000110	1	0	0	0	1	1	0
14	1101	1	1	0	1	$p^3 + p^2 + 1$	$p^6 + p^4 + 1$	1010001	1	0	1	0	0	0	1
15	1011	1	0	1	1	$p^3 + p + 1$	$p^6 + p^5 + p^4 + p^3 + p^2 + p + 1$	1111111	1	1	1	1	1	1	1
16	1111	1	1	1	1	$p^3 + p^2 + p + 1$	$p^6 + p^3 + p + 1$	1001011	1	0	0	1	0	1	1

Table 3.1 The $(n = 7,\ k = 4)$ cyclic code of Example 3.2.

Exercise 3.2 : Verify the correctness of Table 3.1, the linearity and the cyclic properties, either by hand or by running the simple code in Matlab as illustrated in the sample case below

Sample case : Take $\mathbf{c}_2$ and $\mathbf{c}_3$ from Table 3.1 and implement the sample following operations for linearity test

$$\mathbf{c}_2 = (0001101)\ ,\ \mathbf{c}_3 = (0011010)$$
$$\mathbf{c}_2 \oplus \mathbf{c}_3 = \begin{array}{l} 0\,0\,0\,1\,1\,0\,1 \\ \underline{0\,0\,1\,1\,0\,1\,0} \\ 0\,0\,1\,0\,1\,1\,1 = \mathbf{c}_6 \end{array} \qquad (3.10)$$

The operation in (3.10) is performed in Matlab by the following code

```
c2 = [0 0 0 1 1 0 1]; % Define c2
```

```
c3 = [0 0 1 1 0 1 0]; % Define c3
rem(c2 + c3, 2); % Perform modulo-2 addition
```

Sample test for the cyclic property with $\mathbf{c}_2$ and $\mathbf{c}_3$

$$\begin{aligned}\mathbf{c}_2^{(0)} &= (0001101)\ ,\ \mathbf{c}_2^{(1)} = (0011010) = \mathbf{c}_3^{(0)}\ ,\ \mathbf{c}_2^{(2)} = (0110100) = \mathbf{c}_4^{(0)} \\ \mathbf{c}_3^{(0)} &= (0011010)\ ,\ \mathbf{c}_3^{(1)} = (0110100) = \mathbf{c}_4^{(0)}\ ,\ \mathbf{c}_3^{(2)} = (1101000) = \mathbf{c}_5^{(0)}\end{aligned} \tag{3.11}$$

The generator matrix $\mathbf{G}$ to arrive at the code words in the last column of Table 3.1, can simply be obtained by placing on the matrix rows the following shifted versions of the generator polynomial $g(p) = p^3 + p^2 + 1$

$$\begin{aligned}&\text{Row 1 of } \mathbf{G}: p^3 g(p) = p^6 + p^5 + p^3 \\ &\text{Row 2 of } \mathbf{G}: p^2 g(p) = p^5 + p^4 + p^2 \\ &\text{Row 3 of } \mathbf{G}: p^1 g(p) = p^4 + p^3 + p \\ &\text{Row 4 of } \mathbf{G}: p^0 g(p) = p^3 + p^2 + 1\end{aligned}$$

$$\begin{matrix} p^6 & p^5 & p^4 & p^3 & p^2 & p^1 & p^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow \text{Row 1} \\ \leftarrow \text{Row 2} \\ \leftarrow \text{Row 3} \\ \leftarrow \text{Row 4} \end{matrix} \tag{3.12}$$

Exercise 3.3 : The m file containing message signal symbols of Table 3.1 and the generator polynomial of (3.12) is Cyccode_ECE587.m. Using this m file, verify that we are able to obtain the code words of Table 3.1. Furthermore verify that linearity and cyclic properties are satisfied for the code given in Table 3.1. In its present form, all code words are generated, the linearity is tested only for $\mathbf{c}_9 \oplus \mathbf{c}_{13}$ and the cyclic property is tested for $\mathbf{c}_9$ only up to three cyclic shifts. You are expected to complete the rest either using the present form of Cyccode_ECE587.m, or modifying it.

3.3 Generator Matrix of Cyclic Codes (in systematic form)

As an alternative to (3.12), we can also formulate a generator matrix in systematic form in a similar manner to (2.18). Below we illustrate this with a example, based on the $(n = 7,\ k = 4)$ cyclic code and its associated generator polynomial $g(p) = p^3 + p^2 + 1$.

Example 3.3 : To find the generator matrix of $g(p) = p^3 + p^2 + 1$in systematic form, we perform the following computations

Take p^{n-1}, i. e., p^6 divide it by $g(p)=p^3+p^2+1$, remainder : p^2+p

$$p^5 \mod p^3+p^2+1=p+1 \quad \leftarrow \text{remainder}$$
$$p^4 \mod p^3+p^2+1=p^2+p+1 \quad \leftarrow \text{remainder}$$
$$p^3 \mod p^3+p^2+1=p^2+1 \quad \leftarrow \text{remainder}$$

Sample computation

$$\begin{array}{l|l} p^6 & \dfrac{p^3+p^2+1}{\overline{)\,p^3+p^2+p}} \\ \hline p^6+p^5+p^3 & \\ \hline 0+p^5+p^3 & \\ \quad p^5+p^4+p^2 & \\ \hline p^4+p^3+p^2 & \\ p^4+p^3+p & \\ \hline \end{array}$$

$$p^2+p \quad \leftarrow \text{ remainder} \tag{3.13}$$

Now we place these remainders in the $(k=4)\times(n-k=3)$ **P** part of $(k=4)\times(n=7)$ **G** (generator matrix) as follows

$$\begin{matrix} & & & & & p^2 & p^1 & p^0 & \leftarrow & \text{remainder} \end{matrix}$$

$$\mathbf{G}=\left[\mathbf{I}_k \middle| \mathbf{P}\right]=\left[\begin{array}{cccc|ccc} 1&0&0&0&1&1&0 \\ 0&1&0&0&0&1&1 \\ 0&0&1&0&1&1&1 \\ 0&0&0&1&1&0&1 \end{array}\right] \;,\; \mathbf{I}_k=\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \;,\; \mathbf{P}=\begin{bmatrix} 1&1&0 \\ 0&1&1 \\ 1&1&1 \\ 1&0&1 \end{bmatrix} \quad \Leftarrow \quad \boxed{\begin{array}{l} p^2+p \\ p+1 \\ p^2+p+1 \\ p^2+1 \end{array}} \tag{3.14}$$

Exercise 3.4 : The systematic generator matrix is included in the m file named Cyccodesys_ECE587.m. Run this file to find the code words, test to see if they are different from those listed in Table 3.1. Test if the code words generated by this systematic generator matrix satisfy the linearity and cyclic properties. Also evaluate the Hamming distance, weights of all code words, the minimum distance and the minimum weight.

4. Convolutional Codes

4.1 Basics

The general block diagram of a convolutional encoder is shown in Fig. 4.1 (pasted from Fig. 9.24 of Proakis 2002).

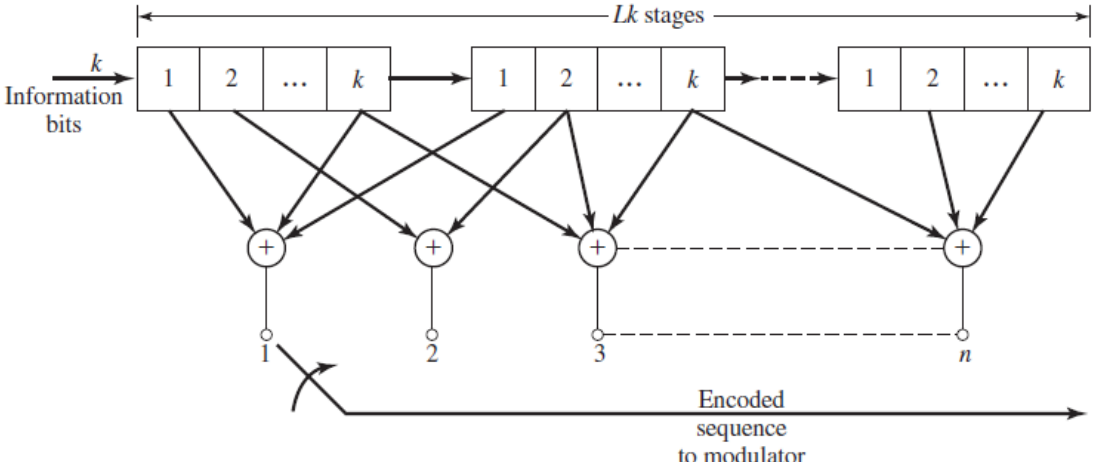


Fig. 4.1 General block diagram of a convolutional encoder.

According to Fig. 4.1, we take as message sequence (binary) information bits of length k at a time and store those in L (also called the constraint length) stages of shift registers. Then from the adders that are connected to these shift registers, we read an output of length n, where k bits are due to the message, thus the extra $n-k$ bits appear because of the coding.

Generally for traceable results, we prefer much reduced configurations than the one depicted in Fig. 4.1. Such an example, based on Example 9.7.1 of Proakis 2002, is given below.

Example 4.1 : For the $(n=2,\ k=1)$ convolutional encoder shown below, find the output, if the input message sequence is $\mathbf{x}=[1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1]$

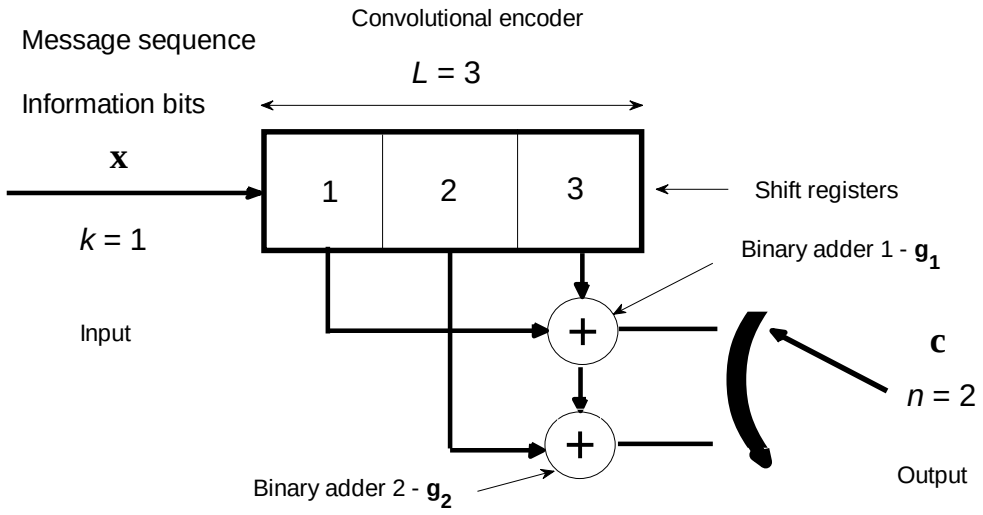


Fig. 4.2 Convulutional encoder for Example 4.1.

Solution for Example 4.1 : We can find the output in three ways,

a) By noting the contents of shift registers at successive loadings from the input side and performing the act of adding at the binary adders by observing the binary addition rules and finally scanning the output of the adders in the shown direction (downwards). Before the first binary entry (which is the leftmost binary value in $\mathbf{x}$), we assume shift registers 1 and 2 are loaded with zeros. In fact, to create the same conditions for the next entry, we append two zeros to the end of the existing input message such that our actual input becomes $\mathbf{x}=[1,\ 1,\ 0, 1,\ 0, 1,\ 1, 0, 0]$. Then by manual tracing we find the output from the convolutional encoder of Fig. 4.2 is $\mathbf{c}=[1,\ 1,\ 1, 0, 1,\ 0, 0, 0, 0, 1,\ 0, 0, 1, 0, 1, 0, 1, 1]$. Comparing $\mathbf{c}$ with $\mathbf{x}$, we see no resemblance between the two. This means, for a decoder that has no knowledge of the encoder drawn in Fig. 4.2, it is almost impossible to go back from $\mathbf{c}$ to $\mathbf{x}$.
b) The output of the convolutional encoder can also b found by interlacing method. For this, we write the polynomial representation of the adders connections to the shift registers as follows

$$\begin{aligned}\mathbf{g}_1&=[1,\ 0, 1], \qquad \mathbf{g}_2=[1,\ 1, 1]\\ \mathbf{g}_1(p)&=p^2+1,\ \mathbf{g}_2(p)=p^2+p+1\end{aligned} \tag{4.1}$$

As apparent from (4.1), for the $\mathbf{g}$ representation, if there is connection of the adder to the shift register, we write one, otherwise we write zero. Then we convert this representation to $\mathbf{g}(p)$ polynomial expressions as shown on the second line of (4.1). Using the same convention, we write the input message sequence (without the added zeros at the end) as a polynomial expression as follows

$$\begin{aligned}&\qquad 1\times p^6, 1\times p^5,\ 0\times p^4,\ 1\times p^3,\ 0\times p^2,\ 1\times p^1,\ 1\times p^0\\ \mathbf{x}=&\ [1, \quad 1, \quad 0, \quad 1, \quad 0, \quad 1, \quad 1]\\ \mathbf{x}(p)&=p^6+p^5+p^3+p+1\end{aligned} \tag{4.2}$$

Now by observing binary addition rules, we proceed as follows

$$\begin{aligned}\mathbf{x}(p)\mathbf{g}_1(p)&=(p^6+p^5+p^3+p+1)(p^2+1)=p^8+p^7+p^6+p^2+p+1\\ \mathbf{x}(p)\mathbf{g}_2(p)&=(p^6+p^5+p^3+p+1)(p^2+p+1)=p^8+p^4+1\end{aligned} \tag{4.3}$$

Then the output of the encoder will be obtained by the interlacing $\mathbf{x}(p)\mathbf{g}_1(p)$ with $\mathbf{x}(p)\mathbf{g}_2(p)$ as shown below

$$\begin{aligned}&p^8 \text{ of } \mathbf{x}(p)\mathbf{g}_1(p) \quad \cdots \quad p^4 \text{ of } \mathbf{x}(p)\mathbf{g}_1(p) \quad \cdots\\ &\quad \downarrow \qquad\qquad\qquad \downarrow\\ &\mathbf{c}=[1, \quad 1,\ 1, 0, 1,\ 0, 0, 0, 0, \quad 1,\ 0, 0, 1, 0, 1, 0, 1, 1]\\ &\qquad\ \uparrow \qquad\qquad\qquad \uparrow\\ &p^8 \text{ of } \mathbf{x}(p)\mathbf{g}_2(p) \quad \cdots \quad p^4 \text{ of } \mathbf{x}(p)\mathbf{g}_2(p) \quad \cdots\end{aligned} \tag{4.4}$$

As seen, the output given in (4.4) is in perfect agreement with the one found by hand tracing in a).

c) As a final option, we can find the output of the convolution encoder using Matlab. For this we must introduce the construction of convolutional encoder in Fig. 4.2 to Matlab in the notation of Matlab. This is done by specifying the arguments in the function named "poly2trellis(ConstraintLength, CodeGenerator)". In the configuration of Fig. 4.2, we have one layer of shift registers, since $k=1$. Then the argument, ConstraintLength will be represented by the number of shift registers in the single layer $L=3$. For the CodeGenerator argument, we benefit from $\mathbf{g}_1$ and $\mathbf{g}_2$ given in (4.1) convert those the binary elements there first to decimal then to octal to obtain,

$$\begin{array}{ccc} \mathbf{g}_1=[1,\ 0,\ 1] & & \mathbf{g}_2=[1,\ 1,\ 1] \\ \downarrow & & \downarrow \\ \text{convert 101 to } 1\times 2^2+0\times 2^1+1\times 2^0=5 \text{ - decimal} & & \text{convert 111 to } 1\times 2^2+1\times 2^1+1\times 2^0=7 \text{ - decimal} \\ \text{convert to octal} \quad \downarrow & & \downarrow \\ \text{CodeGenerator} = \quad [5, & & 7] \end{array} \tag{4.5}$$

Thus we can specify the arguments of the function poly2trellis for the encoder configuration of Fig. 4.2 as follows

$$\begin{array}{rccc} \text{Constraint length - } L & \mathbf{g}_1 & \mathbf{g}_2 & \\ \downarrow & \downarrow & \downarrow & \\ \text{t} = \text{poly2trellis}\left(3, \quad [5, \quad 7]\right) & & & \end{array} \tag{4.6}$$

In general, ConstraintLength is $1\times k$ row array, indicating L values for each layer. CodeGenerator is $k\times N_a$ matrix denoting the connections of the adders to the layers of shift registers, where N_a corresponds to the number of adders. Note that in the case $k>1$, in CodeGenerator matrix, the connections of each individual adder each layer of shift registers should be defined one by one. Note further that in Matlab the first shift register is not drawn, but the numeric value of L is defined in the same way as indicated in Fig. 4.2.

Exercise 4.1 : Using the m file Conv12_Exp2.m available on the course webpage, test that the message sequence of $\mathbf{x}=[1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0]$ gives an output (code word) of $\mathbf{c}=[1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1]$. The k to n correspondence between the input and the output is shown below in (4.7) to make tracing easier. Try at least five different message sequences, compare the outputs obtained from Matlab and one of the methods above, i.e. by hand tracing or interlacing.

$$\begin{array}{rccccccccc} \mathbf{x}=[1, & 1, & 0, & 1, & 0, & 1, & 1, & 0, & 0] \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathbf{c}= [\overbrace{1,\ 1}^{1} & \overbrace{1,\ 0}^{1} & \overbrace{1,\ 0}^{0} & \overbrace{0,\ 0}^{1} & \overbrace{0,\ 1}^{0} & \overbrace{0,\ 0}^{1} & \overbrace{1,\ 0}^{1} & \overbrace{1,\ 0}^{0} & \overbrace{1,\ 1}^{0}] \end{array} \tag{4.7}$$

4.2 State and Trellis Diagrams of Convolutional Codes

Since convolutional codes have memory, i.e. dependence on the past, it is customary to model the encoding in the form of state transition diagrams and trellis diagrams.

In Fig. 4.3, we show the state transition diagram for the encoder of Fig. 4.2 (copied directly from Fig. 9.26 of Proakis 2002). In Fig. 4.3, the square boxes represent the contents of the first and second shift registers of Fig. 4.2, therefore named the state. The curved arrows coming out of the state boxes show the transitions from one state to the other together with the input (entering into the first shift register on the left hand side in Fig. 4.2) and the corresponding output in the format of input (bit) / output (bit). It is possible to verify the outputs shown on the transition paths of Fig. 4.3 using the Matlab facilities. For this, the convolutional encoder must be put in the correct state. Assume that we want to test that when the encoder shift registers (first and second) are in state 01, then a transition to 10 will occur and the output will be 00, if we give an input of 1. To do this, we may use the m file called "Conv12_Exp2.m" available on the course webpage. In this m file, the encoder configuration of Fig. 4.2 is already defined in the form of (4.6). Bearing in mind that initially, the three registers (of Fig. 4.2) is loaded with all zeros, the test of transition 01 to 10 will take place, if we give the input message sequence of $\mathbf{x} = [1,\ 0,\ 1]$. This way, we will get the following in the Matlab workspace

	Message =	1	0	1	$\leftarrow \mathbf{x}$ (input)
	Code =	$\overbrace{0\quad 1}^{1}$	$\overbrace{1\quad 1}^{0}$	$\overbrace{0\quad 0}^{1}$	$\leftarrow \mathbf{c}$ (output)
Contents of shift register		$\underbrace{0\,0\,0}_{\text{initial loading}}$	$\underbrace{1\,0\,0}_{1}$	$\underbrace{0\,1\,0}_{0}$	$\underbrace{1\,0\,0}_{1}$
States		$\underbrace{0\ 0}_{\text{initial loading}}$	$\underbrace{1\ 0}_{1}$	$\underbrace{0\ 1}_{0}$	$\underbrace{1\ 0}_{1}$

(4.8)

From (4.8), we verify that the convolutional encoder of Fig. 4.2, indeed gives an output of 00, when going from state 01, to a state 10 with an input of 1.

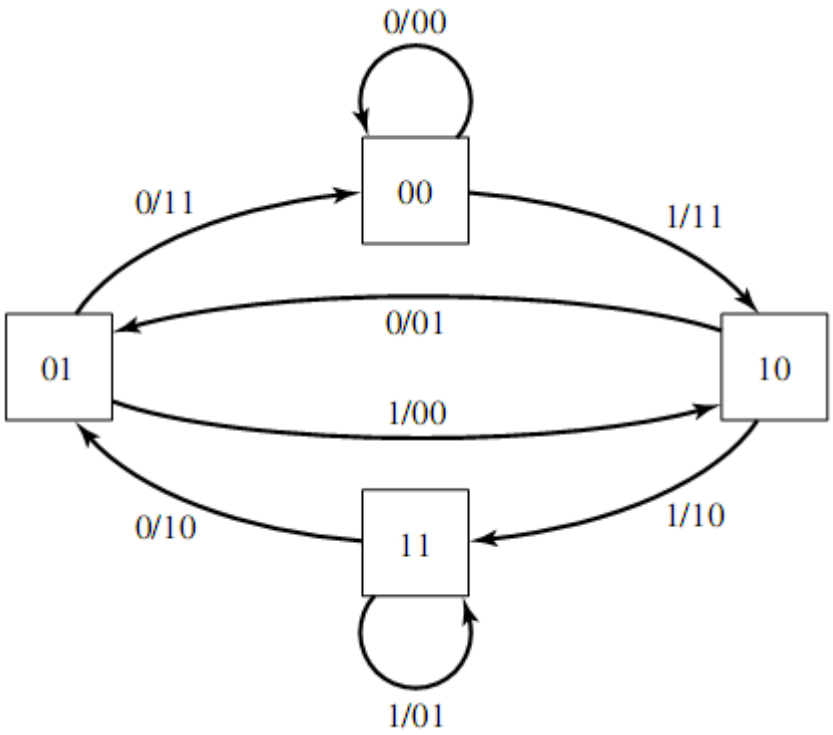


Fig. 4.3 State transition diagram of the convolutional encoder of Fig. 4.2.

Exercise 4.2 : Using the m file Conv12_Exp2.m available on the course webpage, verify that that the transitions between the other states of Fig. 4.3 occur at the given inputs, yielding the indicated outputs.

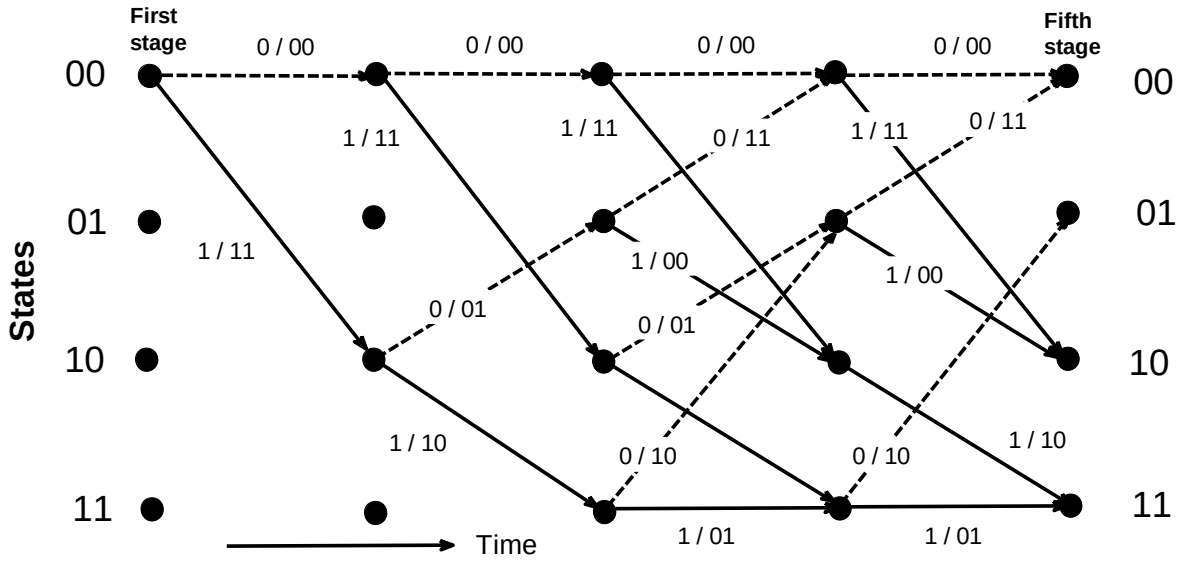


Fig. 4.4 Trellis diagram of the encoder in Fig. 4.2.

Of course, the state transition diagram is not a convenient tool for tracing the output along time axis. To serve such a need trellis diagram is developed. The trellis diagram of the encoder of Fig. 4.2 is given in Fig. 4.4. If required, the trellis diagram of Fig. 4.4 can be expanded to the right to any number of stages depending on the length of the encoder output. Similar to the state transition diagram, the trellis diagram also contains (transition) paths with input (bit) / output

(bit) indicated on them, where we have used solid line arrows for an input of 1, broken line arrows for an input of 0.

Exercise 4.3 : Using the state transition diagram in Fig 4.3 and the trellis diagram of Fig. 4.4, test that the message sequence of $\mathbf{x} = [1,\ 1,\ 0, 1,\ 0, 1,\ 1, 0, 0]$ gives an output (codeword) of $\mathbf{c} = [1,\ 1,\ 1, 0, 1,\ 0, 0, 0, 0, 1,\ 0, 0, 1, 0, 1, 0, 1, 1]$.

4.3 Decoding Convolutional Codes

Decoding a convolutionally encoded codeword to retrieve the message sequence requires the use of Veterbi algorithm. The basic steps are summarized as follows ;

a) Take the received sequence and parse it into subsequences at (bit) lengths of n and distribute them above the paths connecting all zero states.
b) Draw a trellis diagram (with paths) with a number of stages equal to the number of parsed subsequences. For the last $L-1$ stages, draw only paths corresponding to all zero inputs.
c) Start on the left hand side at the all zero state; by also setting the accumulated distance metrics to zero.
d) Make advances to the right and upon reaching at a state of some stage via different branches, identify the paths that yield the minimum distance and name those as survivors. Delete the immediate prior paths that that do not satisfy this minimum distance criteria.
e) Continue in this manner till reaching the all zero state of the final stage.
f) Mark the full path that starts from the all zero state and end at all zero state and has that the minimum accumulated distance metric value as the chosen path, retrieve the information sequence from the input values on the individual links of that path.

Example 4.2 : The encoder of Fig. 4.2 produces some code word and after being transmitted it is received as $\mathbf{y} = [0,\ 1,\ 1, 0, 1,\ 1, 1, 1, 0, 1,\ 0, 0, 0, 1]$. Decode this sequence to find the information sequence.

Solution to Example 4.2 : We implement the Veterbi algorithm steps a to e stated above and obtain the trellis diagram given in Fig. 4.5.

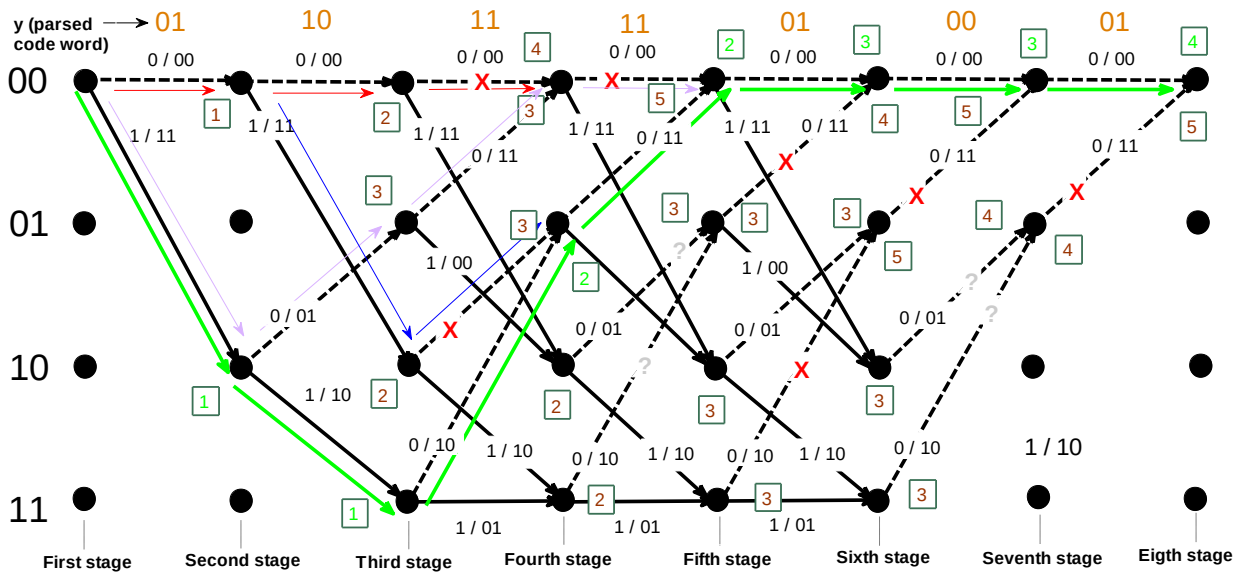


Fig. 4.5 The trellis diagram for decoding the code word **y** given in Example 4.2.

From Fig. 4.5, we deduce that the path followed by the green arrows is the path that gives the minimum accumulated metrics. Its progress through the trellis diagram is tabulated below in Table 4.1.

Parsed code word	Path input / output	Originating state	Terminating state	Accumulated metric value
01	1 / 11	State **00** of First stage	State **10** of Second stage	1
10	1 / 10	State **10** of Second stage	State **11** of Third stage	1
11	0 / 10	State **11** of Third stage	State **01** of Fourth stage	2
11	0 / 11	State **01** of Fourth stage	State **00** of Fifth stage	2
01	0 / 00	State **00** of Fifth stage	State **00** of Sixth stage	3
00	0 / 00	State **00** of Sixth stage	State **00** of Seventh stage	3
01	0 / 00	State **00** of Seventh stage	State **00** of Eighth stage	4

Table 4.1 The tabulation of the path that will give the minimum accumulated metric value for the trellis diagram in Fig. 4.5.

From the inputs of the paths in the second column of Table 4.1, we obtain the message as $\mathbf{x} = [1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0]$, where the two zeros at the very end of the right hand side are those used to flush the register contents to zero, making it possible to start from all zero state for the next message signal. The value of accumulated metric value being 4 indicates that four bits in the decoded message are in error. In Fig. 4.5, all possible paths that link the state 00 of the first stage to the state 00 of the final (eighth) stage are shown. The ones marked with **X** are the nonsurviving paths and are deleted because of the requirement set in item d given in the basic steps of the Veterbi algorithm. In some cases we arrive at the some state of certain stage with an equal accumulated metric value. Such examples are state 01 of the fifth and seventh stages. This issue is resolved by advancing further to next stages.

Exercise 4.4 : Verify that the path tracing shown in Fig. 4.5 is correct according to the Veterbi algorithm steps.

Exercise 4.5 : Using the m file Conv12_Exp2.m available on the course webpage, verify that that given an received code word sequence of $\mathbf{y} = [0,\ 1,\ 1,\ 0,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1]$ and the encoder configuration of Fig. 4.2, we get the message sequence as $\mathbf{x} = [1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0]$.

Modifying the configuration given in Fig. 4.2 slightly, it is possible to arrive at the convolutional encoder of $(n=3,\ k=1)$

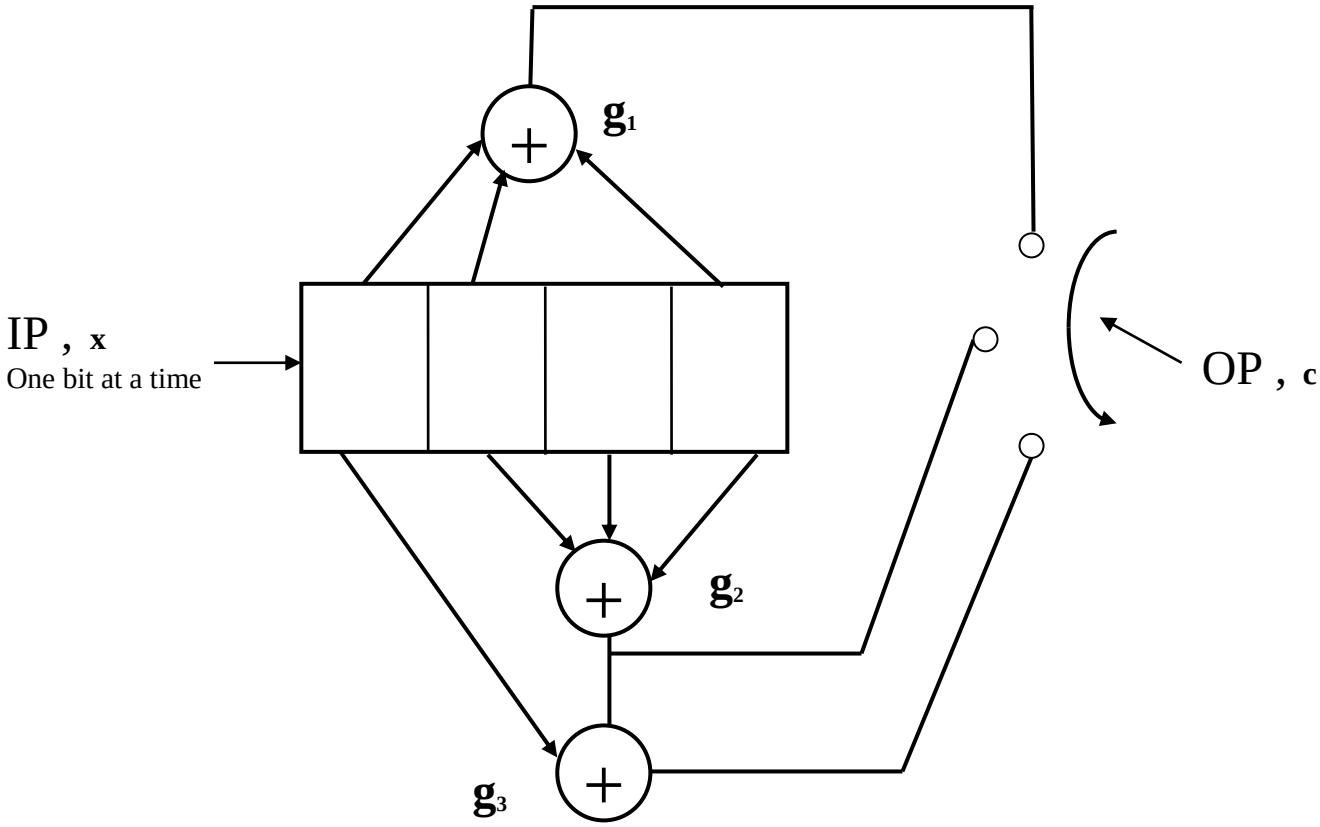


Fig. 4.6 A $(n=3,\ k=1)$ convolutional encoder.

In the encoder of Fig. 4.6, there is four shift registers, i.e., the constraint length $L=4$. Since we feed input one bit at a time, i.e., $k=1$, then our dependence on the past is three bits. This way our state machine will consist of eight states, as shown in Fig. 4.7 (copied from the handwritten solutions of ECE 587MT_21112011_Solutions).

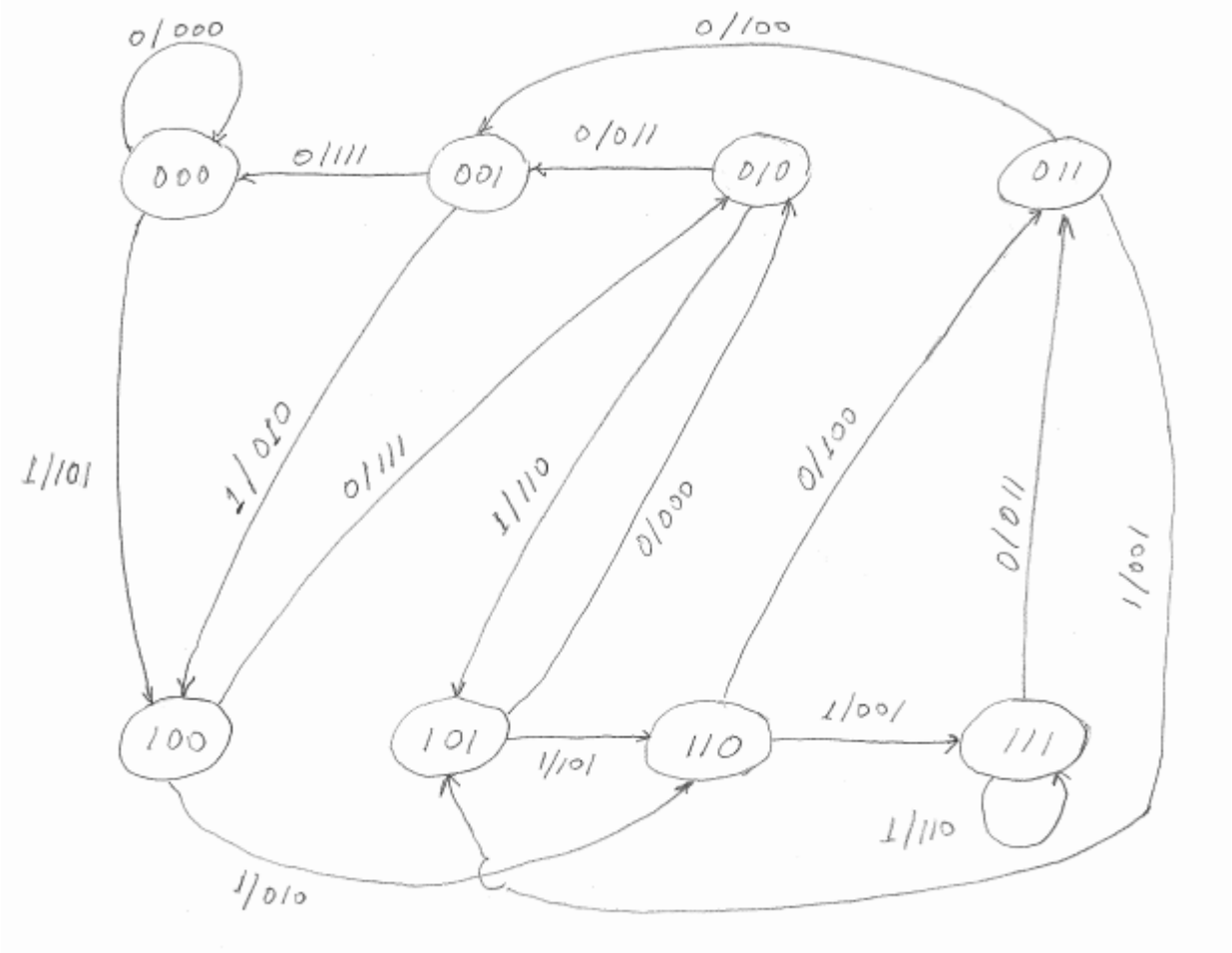


Fig. 4.7 State transition diagram of the convolutional encoder given in Fig. 4.6.

By following the procedure outlined in (4.5) and (4.6), we get

$$\mathbf{g}_1 = [1,\ 1,\ 0,\ 1] \qquad \mathbf{g}_2 = [0,\ 1,\ 1,\ 1]$$

$$1101 \text{ to } 1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0 = 13 \qquad 0111 \text{ to } 0\times 2^3+1\times 2^2+1\times 2^1+1\times 2^0 = 7$$

convert to octal

$$\text{CodeGenerator} = [15,\quad 7,\quad 17]$$

convert to octal

$$1111 \text{ to } 1\times 2^3+1\times 2^2+1\times 2^1+1\times 2^0 = 15$$

$$\mathbf{g}_3 = [1,\ 1,\ 1,\ 1] \tag{4.9}$$

Thus we can specify the arguments of the function poly2trellis for the encoder configuration of Fig. 4.6 as follows

$$\text{t} = \text{poly2trellis}\left(4,\ [15,\ 7,\ 17]\right) \tag{4.10}$$

(Constraint length - L; $\mathbf{g}_1$, $\mathbf{g}_2$, $\mathbf{g}_3$)

Now by setting the input message sequence as $\mathbf{x}=[1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,1]$ and using Conv12_Exp2.m with the poly2trellis function defined in (4.10), we get $\mathbf{c}=[1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1]$.

Exercise 4.6 : Using the m file Conv12_Exp2.m available on the course webpage, verify that that an input of $\mathbf{x}=[1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,1]$, and the poly2trellis definition of (4.10), we get a code word of $\mathbf{c}=[1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1]$. Further decode this code word to get back to the original message. Again using Conv12_Exp2.m, verify the state transition diagram in Fig. 4.7. Construct the trellis diagram of the encoder in Fig. 4.6.

We defer the probability of error calculations for convolutionally encoded messages for later. Next, we introduce an efficient use of convolutional encoding in Trellis Coded Modulation (TCM)

5. Trellis Coded Modulation (TCM)

TCM was first proposed by Ungerboeck in 1981. The basic idea is to get better performance in PSK and QAM via the use of convolutional coding.

Trellis coded modulation can best be described by the following simplest example. Assume that we have a message signal that we can group as binary waveforms (bits) to make 4 PSK. Here instead of converting our message signal to 4 PSK symbols, we take the binary form of the message grouped as two bits and use convolutional encoding to generate three bits per symbol. Thus in coding notation, $k=2,\ n=3$. Such an operation can be performed by the encoder shown below in Fig. 5.1

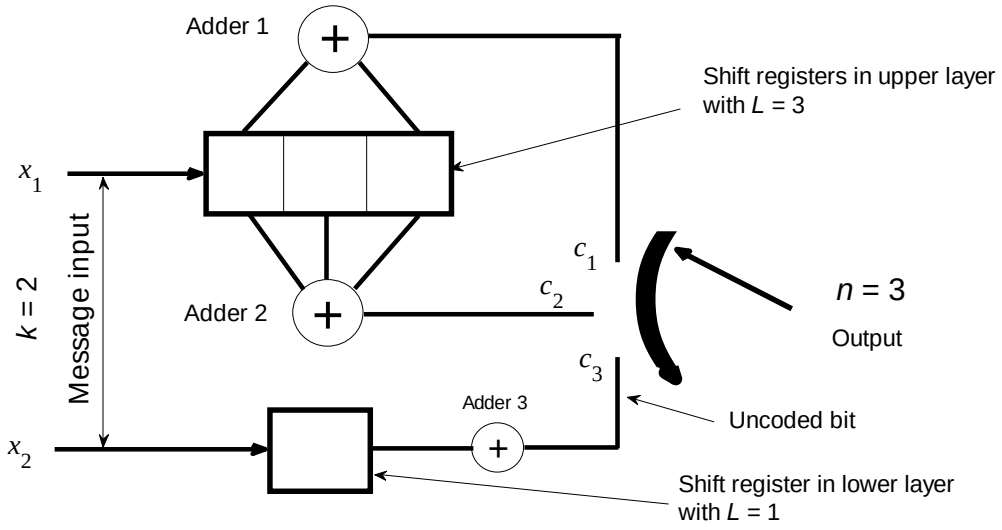


Fig. 5.1 Convolutional encoder for 8 PSK TCM.

Note that in order to achieve a coding rate of $k/n=2/3$, in Fig. 5.1, we have had to encode only one bit (upper bit) of the message and let the other bit (lower) bit pass uncoded. The encoder configuration of Fig. 5.1 must be introduced to Matlab in its entirety, i.e. including the

lower layer that illustrates that no coding is to be performed on the message input bit x_2. From the guidelines given in (4.6) and (4.10), it is easy to find that the encoder configuration in Fig. 5.1 can be defined in Matlab notation as

$$\text{poly2trellis(ConstraintLength, CodeGenerator)} = \text{poly2trellis}\left([3,\ 1],\ [5,\ 7,\ 0;0,\ 0,\ 1]\right) \qquad (5.1)$$

Now we come to the placement of the symbols obtained as a result of the coding operation of Fig. 5.1in a constellation. Assuming that PSK is selected for this purpose, then it is easy to guess that 8 PSK would have to be utilized in order to accommodate the three bit grouping (i.e. eight level signalling). The ordering of symbols, i.e. signal vectors, also called mapping, in an (conventional, uncoded) 8 PSK constellation can be in two ways as depicted in Fig. 5.2.

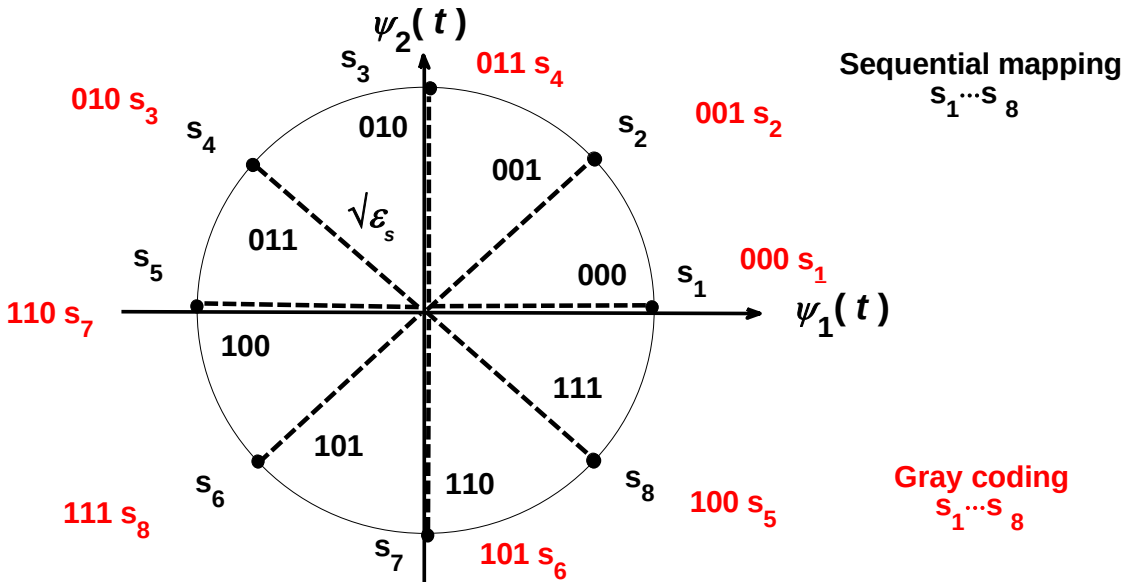


Fig. 5.2 Two ways of mapping the three bits to 8 PSK symbols.

As seen from Fig 5.2, the ordering on the inner side constellation circle is simply the sequential mapping (in ascending numeric order) of symbols from (level) 0 to (level) 7. Taking into account that most likely errors come from one transmitted signal being incorrectly detected as the adjacent signals, it is possible to arrange the mapping of bits to symbols as those placed outside the constellation circle (in red). The idea in the latter mapping is that the adjacent symbols should differ only by one bit, resulting in most errors being bit errors, rather than symbol errors. Such an arrangement is known as Gray coding. Although Gray coding brings some improvement over the simple sequential ordering of symbols around constellation circle, a substantial increase can only be achieved by taking into account the property embedded into TCM via coding.

To arrive at 8 PSK TCM constellation diagram, we must first construct the state trellis diagram for the encoder illustrated in Fig. 5.1. This is done by either hand tracing or using Matlab and the result is shown below in Fig. 5.3. Note that in Fig. 5.3, only two stages are shown (hence time unfolding is suppressed), therefore unlike Fig. 4.5, we do not start at all zero state. So Fig. 5.3 is somewhat similar to the state transition diagram of Fig. 4.3.

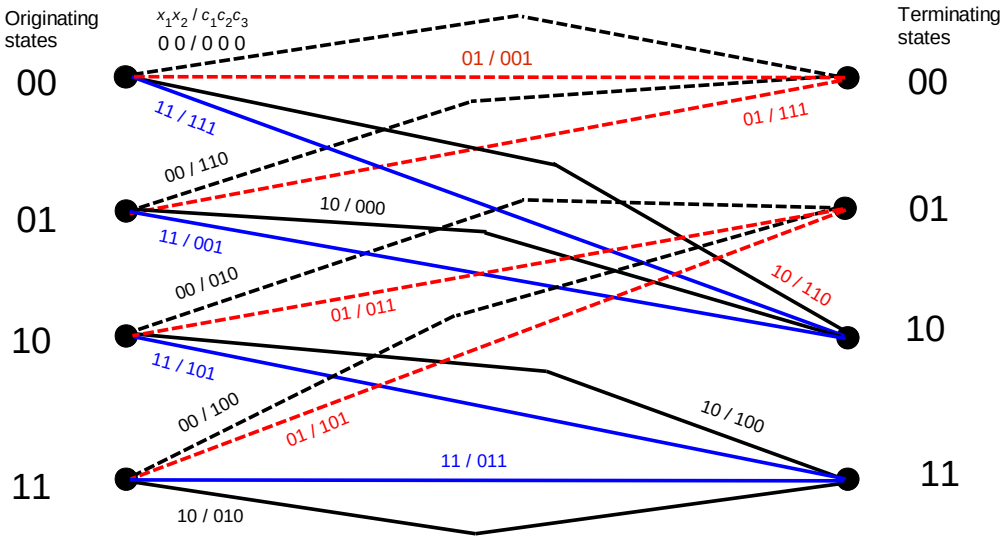


Fig. 5.3 State trellis diagram for the convolutional encoder of Fig. 5.1.

We know that in convolutional coding, the states are determined by the contents of the shift registers which contain bits from the past inputs. In the configuration of Fig. 5.1, this is applicable to the shift registers in the upper layer, since the lower layer does not perform any coding. This way, our state remains the same against the changes in the second bit (i.e. x_2). This phenomena leads to the emergence of parallel paths, consisting of two links. Such parallel paths start from the same originating state and merge into the same terminating state, as demonstrated by Fig. 5.3. On the paths of Fig. 5.3, the related two input bits and three output bits are also indicated with a separator sign of " / ".

Exercise 5. 1 : Verify Fig. 5.3 either by hand tracing in Fig. 5.1 or by modifying the m file Conv12_Exp2.m according to (5.1). Note that if an input of $\mathbf{x} = [0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1]$ is used, then the full state trellis diagram of Fig. 5. 3 is generated.

In order to arrive at TCM constellation, it is instructive to organize the information offered in Fig. 5.3 in the form of two tables, one for the originating states, the other for the terminating states. These are listed below.

Originating state	Output symbol	**s** designation	**B** designation	Parallel paths
00	000	$\mathbf{s}_1$	$\mathbf{B}_1$	Parallel
00	001	$\mathbf{s}_2$	$\mathbf{B}_1$	
00	110	$\mathbf{s}_7$	$\mathbf{B}_1$	Parallel
00	111	$\mathbf{s}_8$	$\mathbf{B}_1$	
01	110	$\mathbf{s}_7$	$\mathbf{B}_1$	Parallel
01	111	$\mathbf{s}_8$	$\mathbf{B}_1$	
01	000	$\mathbf{s}_1$	$\mathbf{B}_1$	Parallel
01	001	$\mathbf{s}_2$	$\mathbf{B}_1$	

10	011	$\mathbf{s}_4$	$\mathbf{B}_2$	Parallel
10	010	$\mathbf{s}_3$	$\mathbf{B}_2$	
10	100	$\mathbf{s}_5$	$\mathbf{B}_2$	Parallel
10	101	$\mathbf{s}_6$	$\mathbf{B}_2$	
11	100	$\mathbf{s}_5$	$\mathbf{B}_2$	Parallel
11	101	$\mathbf{s}_6$	$\mathbf{B}_2$	
11	010	$\mathbf{s}_3$	$\mathbf{B}_2$	Parallel
11	011	$\mathbf{s}_4$	$\mathbf{B}_2$	

Terminating state	Output symbol	**s** designation	B designation	Parallel paths
00	000	$\mathbf{s}_1$	$\mathbf{B}_1$	Parallel
00	001	$\mathbf{s}_2$	$\mathbf{B}_1$	
00	110	$\mathbf{s}_7$	$\mathbf{B}_1$	Parallel
00	111	$\mathbf{s}_8$	$\mathbf{B}_1$	
01	010	$\mathbf{s}_3$	$\mathbf{B}_2$	Parallel
01	011	$\mathbf{s}_4$	$\mathbf{B}_2$	
01	100	$\mathbf{s}_5$	$\mathbf{B}_2$	Parallel
01	101	$\mathbf{s}_6$	$\mathbf{B}_2$	
10	110	$\mathbf{s}_7$	$\mathbf{B}_1$	Parallel
10	111	$\mathbf{s}_8$	$\mathbf{B}_1$	
10	000	$\mathbf{s}_1$	$\mathbf{B}_1$	Parallel
10	001	$\mathbf{s}_2$	$\mathbf{B}_1$	
11	100	$\mathbf{s}_5$	$\mathbf{B}_2$	Parallel
11	101	$\mathbf{s}_6$	$\mathbf{B}_2$	
11	010	$\mathbf{s}_3$	$\mathbf{B}_2$	Parallel
11	011	$\mathbf{s}_4$	$\mathbf{B}_2$	

Table 5.1 Organization of the information in Fig. 5.3 in the form of two tables.

Table 5.1 as illustrates that the output symbols (signal vectors, $\mathbf{s}_1 \cdots \mathbf{s}_8$) falling into the same **B** designations are

$$\mathbf{B}_1 = (000,\ 001,\ 110,\ 111) \quad , \quad \mathbf{B}_2 = (010,\ 011,\ 100,\ 101)$$
$$\mathbf{B}_1 = (\ \mathbf{s}_1,\ \mathbf{s}_2,\ \mathbf{s}_7,\ \mathbf{s}_8) \quad , \quad \mathbf{B}_2 = (\ \mathbf{s}_3,\ \mathbf{s}_4,\ \mathbf{s}_5,\ \mathbf{s}_6) \qquad (5.2)$$

It is observed from Table 5.1 (and from Fig. 5.3) that for a given originating or a terminating state, the paths carry the symbols either from $\mathbf{B}_1$ or $\mathbf{B}_2$. We now assign the signal vectors contained in $\mathbf{B}_1$ and $\mathbf{B}_2$ separately to the vertices of the two squares oriented at 45^0 with respect to each other. In this assignment, no mixing of signal vectors between $\mathbf{B}_1$ and $\mathbf{B}_2$ is allowed. Furthermore, in the diagonal positions of vertices of the squares, the output symbols which belong to the parallel paths are placed. Eventually, these two squares will yield the 8 PSK TCM constellation as shown in Fig. 5.4.

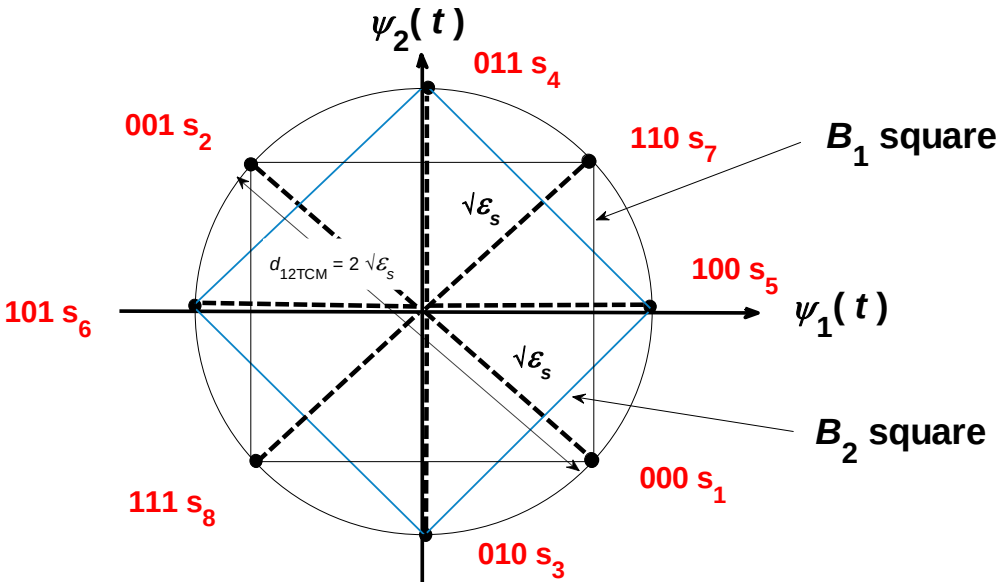


Fig. 5.4 8 PSK TCM constellation

Note that diagonal exchanges between the symbols of the same square (within the same **B** designation) has no effect on the performance as will be shown later in our Matlab runs. Note also that Fig. 9.48 on page 652 of Proakis 2002 does not agree with Fig. 5.4. The rules applied when drawing up the constellation of Fig. 5.4 are named as set partitioning rules by Ungerboeck.

Now we are in a position to compare 8 PSK TCM and 4 PSK in terms of probability of error performance. For high signal to noise ratio (SNR) settings, such a comparison can be quite safely be based on the consideration of the increase in the minimum distance by going from 4 PSK to 8 PSK TCM. In TCM, an error will be made by the routes that will divert from the same state and later merge at the same state. By examining Fig. 3.3, we see that the shortest of such routes are parallel paths. From Fig. 3.4, it is found that these parallel paths (all) have the same distance of $d_{12\mathrm{TCM}} = 2\sqrt{\varepsilon_s}$. Thus $d_{12\mathrm{TCM}} = 2\sqrt{\varepsilon_s}$ will be the minimum distance to be encountered between the symbols of 8 PSK TCM constellation. On the equal average energy basis, if we compare minimum distance of 8 PSK TCM to the minimum distance 4 PSK (that is the distance between black $\mathbf{s}_1$ and $\mathbf{s}_3$), and also relate this to probability of error formulations (denoted by P), we find that

$$d_{\min 8\mathrm{PSKTCM}} = d_{12\mathrm{TCM}} = 2\sqrt{\varepsilon_s} \quad , \quad d_{\min 4\mathrm{PSK}} = d_{12} = \sqrt{2\varepsilon_s} \quad , \quad \frac{d_{\min 8\mathrm{PSKTCM}}}{d_{\min 4\mathrm{PSK}}} = \sqrt{2}$$

$$P_{8\mathrm{PSKTCM}} \propto Q\left(d_{\min 8\mathrm{PSKTCM}}\right) \quad , \quad P_{4\mathrm{PSK}} \propto Q\left(d_{\min 4\mathrm{PSK}}\right) \quad , \quad P_{8\mathrm{PSKTCM}} < P_{4\mathrm{PSK}} \qquad (5.3)$$

The comparison in (5.3) means that under high SNR conditions, a 8 PSK TCM system having the signal energies of $1/\sqrt{2}$ times that of 4 PSK will have equivalent probability of error performance, or 8 PSK TCM will achieve the same probability of error performance as that of 4 PSK with less energy.

Next we examine a more sophisticated case of converting uncoded 8 PSK to 16 QAM TCM. For this, we take the middle encoder of shown Fig. 9 of Ungerboeck's paper [4], which is given below in the notation of these notes and in the configuration to be submitted to Matlab.

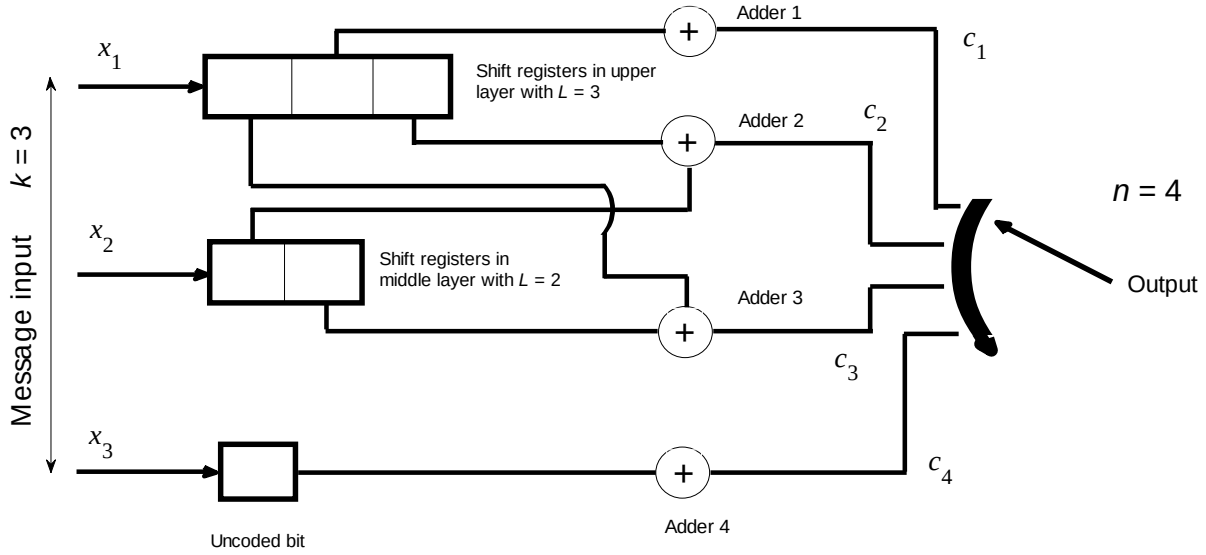


Fig. 5.5 Convolutional encoder for 16 QAM TCM.

By following the steps described above for 8 PSK TCM case, poly2trellis Matlab function for the encoder of Fig. 5.5 can be written as

$$\text{poly2trellis}\left([3,\ 2,\ 1],\ [2,\ 1,\ 4,\ 0;0,\ 2,\ 1,\ 0;0,\ 0,\ 0,\ 1]\right) \tag{5.4}$$

By hand tracing on Fig. 5.5 or by using (5.4) in Matlab, the following state trellis diagram is constructed.

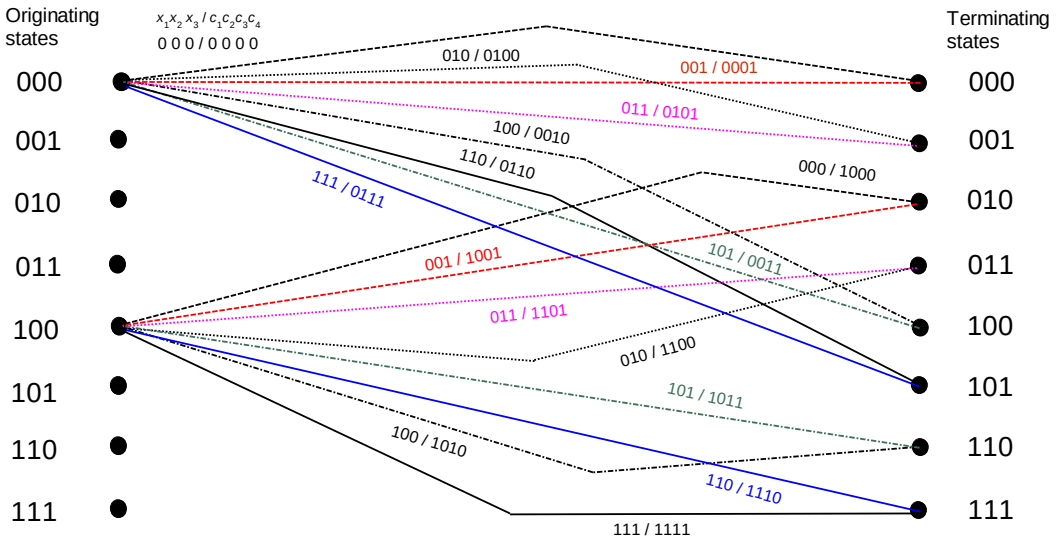


Fig. 5.6 State trellis diagram for the convolutional encoder of Fig. 5.5.

Exercise 5. 2 : Verify the shown paths in Fig. 5.6 either by hand tracing in Fig. 5.5 or by modifying the m file Conv12_Exp2.m according to (5.5).

In Fig. 5.6, due to space limitation, not all details are given. So we construct a table similar to Table 5.1, displaying the outgoing and terminating states (in summarized fashion to save space) separately coupled with **C** and **D** designations.

Originating state	Input/Output symbol	**s** designation	**C** designation	**D** designation	Parallel paths
000	000/0000	$\mathbf{s}_1$	$\mathbf{C}_1$	$\mathbf{D}_1$	Parallel
000	001/0001	$\mathbf{s}_2$	$\mathbf{C}_1$	$\mathbf{D}_1$	
000	010/0100	$\mathbf{s}_5$	$\mathbf{C}_1$	$\mathbf{D}_2$	Parallel
000	011/0101	$\mathbf{s}_6$	$\mathbf{C}_1$	$\mathbf{D}_2$	
000	100/0010	$\mathbf{s}_3$	$\mathbf{C}_2$	$\mathbf{D}_3$	Parallel
000	101/0011	$\mathbf{s}_4$	$\mathbf{C}_2$	$\mathbf{D}_3$	
000	110/0110	$\mathbf{s}_7$	$\mathbf{C}_2$	$\mathbf{D}_4$	Parallel
000	111/0111	$\mathbf{s}_8$	$\mathbf{C}_2$	$\mathbf{D}_4$	
001	000/0010	$\mathbf{s}_3$	$\mathbf{C}_2$	$\mathbf{D}_3$	Parallel
001	001/0011	$\mathbf{s}_4$	$\mathbf{C}_2$	$\mathbf{D}_3$	
001	010/0110	$\mathbf{s}_7$	$\mathbf{C}_2$	$\mathbf{D}_4$	Parallel
001	011/0111	$\mathbf{s}_8$	$\mathbf{C}_2$	$\mathbf{D}_4$	
001	100/0000	$\mathbf{s}_1$	$\mathbf{C}_1$	$\mathbf{D}_1$	Parallel
001	101/0001	$\mathbf{s}_2$	$\mathbf{C}_1$	$\mathbf{D}_1$	
001	110/0100	$\mathbf{s}_5$	$\mathbf{C}_1$	$\mathbf{D}_2$	Parallel
001	111/0101	$\mathbf{s}_6$	$\mathbf{C}_1$	$\mathbf{D}_2$	
010	000/0100	$\mathbf{s}_5$	$\mathbf{C}_1$	$\mathbf{D}_2$	Parallel
010	001/0101	$\mathbf{s}_6$	$\mathbf{C}_1$	$\mathbf{D}_2$	
010	010/0000	$\mathbf{s}_1$	$\mathbf{C}_1$	$\mathbf{D}_1$	Parallel
010	011/0001	$\mathbf{s}_2$	$\mathbf{C}_1$	$\mathbf{D}_1$	
010	100/0110	$\mathbf{s}_7$	$\mathbf{C}_2$	$\mathbf{D}_4$	Parallel
010	101/0111	$\mathbf{s}_8$	$\mathbf{C}_2$	$\mathbf{D}_4$	
010	110/0010	$\mathbf{s}_3$	$\mathbf{C}_2$	$\mathbf{D}_3$	Parallel
010	111/0011	$\mathbf{s}_4$	$\mathbf{C}_2$	$\mathbf{D}_3$	
011	000/0110	$\mathbf{s}_7$	$\mathbf{C}_2$	$\mathbf{D}_4$	Parallel
011	001/0111	$\mathbf{s}_8$	$\mathbf{C}_2$	$\mathbf{D}_4$	

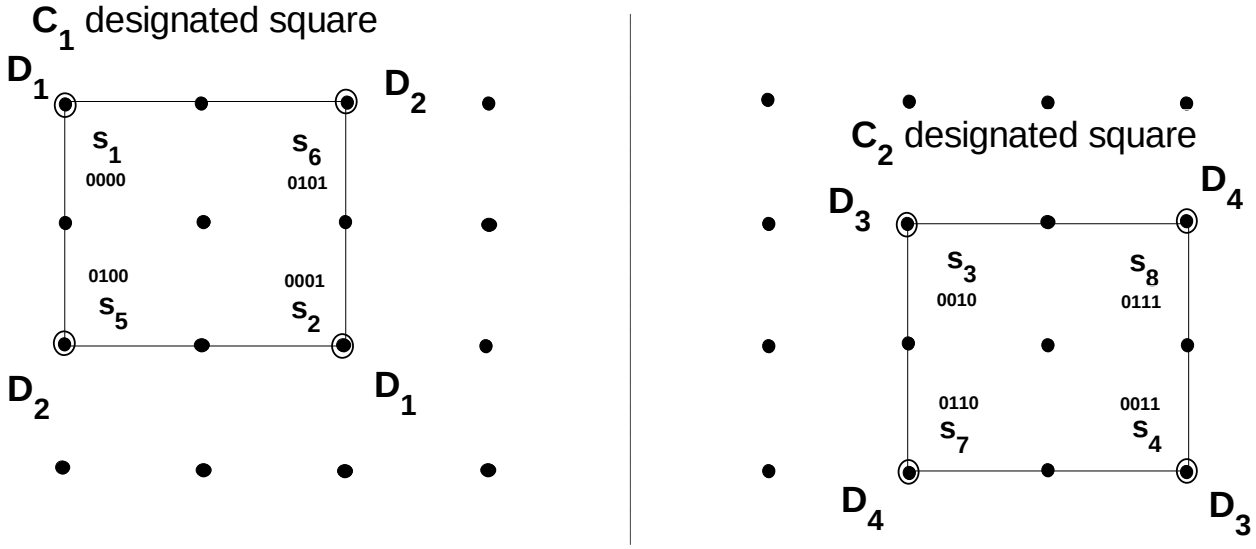
011	010/0010	$\mathbf{s}_3$	$\mathbf{C}_2$	$\mathbf{D}_3$	Parallel
011	011/0011	$\mathbf{s}_4$	$\mathbf{C}_2$	$\mathbf{D}_3$	
011	100/0100	$\mathbf{s}_5$	$\mathbf{C}_1$	$\mathbf{D}_2$	Parallel
011	101/0101	$\mathbf{s}_6$	$\mathbf{C}_1$	$\mathbf{D}_2$	
011	110/0000	$\mathbf{s}_1$	$\mathbf{C}_1$	$\mathbf{D}_1$	Parallel
011	111/0001	$\mathbf{s}_2$	$\mathbf{C}_1$	$\mathbf{D}_1$	
100	000/1000	$\mathbf{s}_9$	$\mathbf{C}_3$	$\mathbf{D}_5$	Parallel
100	001/1001	$\mathbf{s}_{10}$	$\mathbf{C}_3$	$\mathbf{D}_5$	
100	010/1100	$\mathbf{s}_{13}$	$\mathbf{C}_3$	$\mathbf{D}_6$	Parallel
100	011/1101	$\mathbf{s}_{14}$	$\mathbf{C}_3$	$\mathbf{D}_6$	
100	100/1010	$\mathbf{s}_{11}$	$\mathbf{C}_4$	$\mathbf{D}_7$	Parallel
100	101/1011	$\mathbf{s}_{12}$	$\mathbf{C}_4$	$\mathbf{D}_7$	
100	110/1110	$\mathbf{s}_{15}$	$\mathbf{C}_4$	$\mathbf{D}_8$	Parallel
100	111/1111	$\mathbf{s}_{16}$	$\mathbf{C}_4$	$\mathbf{D}_8$	
101	000/1010	$\mathbf{s}_{11}$	$\mathbf{C}_4$	$\mathbf{D}_7$	Parallel
101	001/1011	$\mathbf{s}_{12}$	$\mathbf{C}_4$	$\mathbf{D}_7$	
101	010/1110	$\mathbf{s}_{15}$	$\mathbf{C}_4$	$\mathbf{D}_8$	Parallel
101	011/1111	$\mathbf{s}_{16}$	$\mathbf{C}_4$	$\mathbf{D}_8$	
101	100/1000	$\mathbf{s}_9$	$\mathbf{C}_3$	$\mathbf{D}_5$	Parallel
101	101/1001	$\mathbf{s}_{10}$	$\mathbf{C}_3$	$\mathbf{D}_5$	
101	110/1100	$\mathbf{s}_{13}$	$\mathbf{C}_3$	$\mathbf{D}_6$	Parallel
101	111/1101	$\mathbf{s}_{14}$	$\mathbf{C}_3$	$\mathbf{D}_6$	
110	000/1100	$\mathbf{s}_{13}$	$\mathbf{C}_3$	$\mathbf{D}_6$	Parallel
110	001/1101	$\mathbf{s}_{14}$	$\mathbf{C}_3$	$\mathbf{D}_6$	
110	010/1000	$\mathbf{s}_9$	$\mathbf{C}_3$	$\mathbf{D}_5$	Parallel
110	011/1001	$\mathbf{s}_{10}$	$\mathbf{C}_3$	$\mathbf{D}_5$	
110	100/1110	$\mathbf{s}_{15}$	$\mathbf{C}_4$	$\mathbf{D}_8$	Parallel
110	101/1111	$\mathbf{s}_{16}$	$\mathbf{C}_4$	$\mathbf{D}_8$	
110	110/1010	$\mathbf{s}_{11}$	$\mathbf{C}_4$	$\mathbf{D}_7$	Parallel
110	111/1011	$\mathbf{s}_{12}$	$\mathbf{C}_4$	$\mathbf{D}_7$	
111	000/1110	$\mathbf{s}_{15}$	$\mathbf{C}_4$	$\mathbf{D}_8$	Parallel
111	001/1111	$\mathbf{s}_{16}$	$\mathbf{C}_4$	$\mathbf{D}_8$	
111	010/1010	$\mathbf{s}_{11}$	$\mathbf{C}_4$	$\mathbf{D}_7$	Parallel
111	011/1011	$\mathbf{s}_{12}$	$\mathbf{C}_4$	$\mathbf{D}_7$	

111	100/1100	$\mathbf{s}_{13}$	$\mathbf{C}_3$	$\mathbf{D}_6$	Parallel
111	101/1101	$\mathbf{s}_{14}$	$\mathbf{C}_3$	$\mathbf{D}_6$	
111	110/1000	$\mathbf{s}_9$	$\mathbf{C}_3$	$\mathbf{D}_5$	Parallel
111	111/1001	$\mathbf{s}_{10}$	$\mathbf{C}_3$	$\mathbf{D}_5$	

Originating state	Terminating paths, outputs
000	$\mathbf{C}_1, \mathbf{C}_2$ $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4$ $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_7, \mathbf{s}_8$
001	$\mathbf{C}_1, \mathbf{C}_2$ $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4$ $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_7, \mathbf{s}_8$
010	$\mathbf{C}_3, \mathbf{C}_4$ $\mathbf{D}_5, \mathbf{D}_6, \mathbf{D}_7, \mathbf{D}_8$ $\mathbf{s}_9, \mathbf{s}_{10}, \mathbf{s}_{13}, \mathbf{s}_{14}, \mathbf{s}_{11}, \mathbf{s}_{12}, \mathbf{s}_{15}, \mathbf{s}_{16}$
011	$\mathbf{C}_3, \mathbf{C}_4$ $\mathbf{D}_5, \mathbf{D}_6, \mathbf{D}_7, \mathbf{D}_8$ $\mathbf{s}_9, \mathbf{s}_{10}, \mathbf{s}_{13}, \mathbf{s}_{14}, \mathbf{s}_{11}, \mathbf{s}_{12}, \mathbf{s}_{15}, \mathbf{s}_{16}$
100	$\mathbf{C}_1, \mathbf{C}_2$ $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4$ $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_7, \mathbf{s}_8$
101	$\mathbf{C}_1, \mathbf{C}_2$ $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4$ $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_7, \mathbf{s}_8$
110	$\mathbf{C}_3, \mathbf{C}_4$ $\mathbf{D}_5, \mathbf{D}_6, \mathbf{D}_7, \mathbf{D}_8$ $\mathbf{s}_9, \mathbf{s}_{10}, \mathbf{s}_{13}, \mathbf{s}_{14}, \mathbf{s}_{11}, \mathbf{s}_{12}, \mathbf{s}_{15}, \mathbf{s}_{16}$
111	$\mathbf{C}_3, \mathbf{C}_4$ $\mathbf{D}_5, \mathbf{D}_6, \mathbf{D}_7, \mathbf{D}_8$ $\mathbf{s}_9, \mathbf{s}_{10}, \mathbf{s}_{13}, \mathbf{s}_{14}, \mathbf{s}_{11}, \mathbf{s}_{12}, \mathbf{s}_{15}, \mathbf{s}_{16}$

Table 5.2 Organization of the information in Fig. 5.6 in the form of two tables.

From Table 5.2, from the rule of all originating and terminating paths from a state should be on the corners of $\mathbf{C}_1$ and $\mathbf{C}_2$ or $\mathbf{C}_3$ and $\mathbf{C}_4$ designated squares and parallel paths with $\mathbf{D}$ designations must be placed in diagonal positions of the $\mathbf{C}$ designated squares, we get the allocations of the signal vectors in the 16 QAM TCM constellation as depicted in Fig. 5.7.



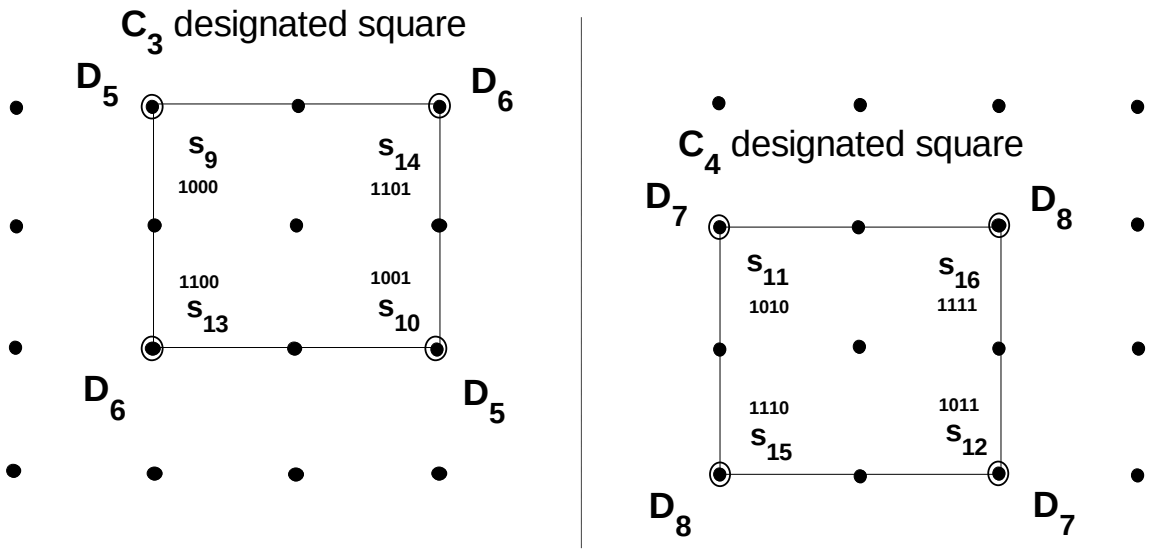


Fig. 5.7 Allocation of signal vectors in 16 QAM TCM constellation based on the information contained in Table 5.2.

In order to compare the performance of 16 QAM TCM against the uncoded 8 PSK, we must equate the average energy in both constellations. This is done by placing the 8 PSK signal vectors on a circle of unity radius adopting the same unity circle for eight signal vectors of 16 QAM TCM and assigning less than unit energy for the most inner ones and more energy for the four outer ones. In the end, the signal vector arrangement of Fig. 5.8 is obtained for 16 QAM TCM.

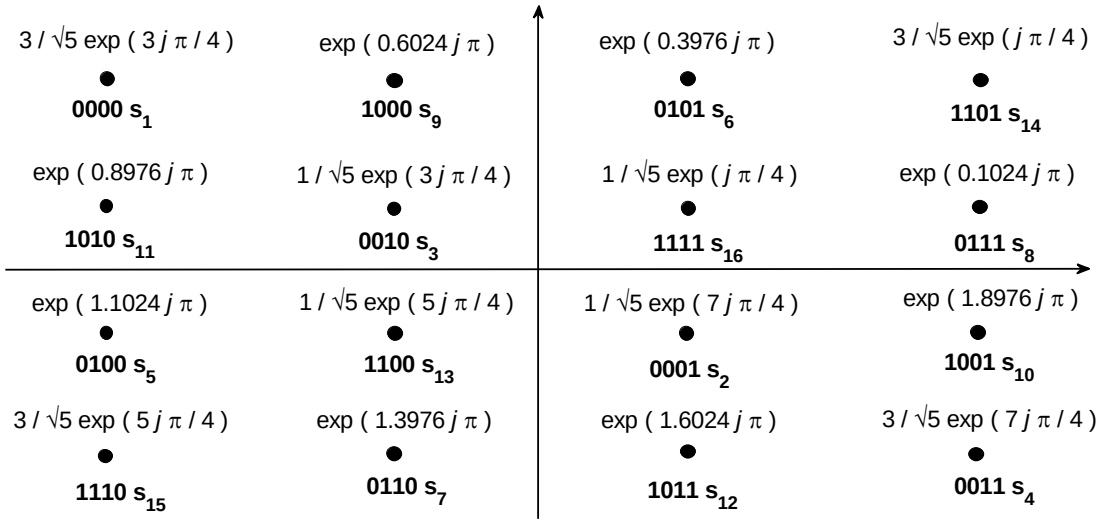


Fig. 5.8 Signal constellation of 16 QAM TCM.

Exercise 5.3 : By running TCM_PeGeneral.m which compares the probability of error performance of 8 PSK TCM against uncoded 4 PSK and TCMPe16QAM.m which compares the probability of error performance of 16 QAM TCM against uncoded 8 PSK using the encoder configurations and constellations of the above, find the relevant probability of error curves. Note the improvements. Verify that performance degrades, when the positions of signal vectors in the

constellations are changed. Use the model files TCM_Dec.mdl and TCM_Enc.mdl during these runs. All Matlab files are available on course webpage.

References

1. John G. Proakis, Masoud Salehi, "Communication Systems Engineering" 2nd Ed., Prentice Hall 2002, ISBN : 0-13-061793-8, Chapter 9.
2. Bernard Sklar, "Digital Communications Fundamentals and Applications", 2nd Ed. Prentice Hall 2002, ISBN : 0-13-084788-7, Chapter 9.
3. John G. Proakis, Masoud Salehi, "Digital Communications" 5th Ed., McGraw Hill 2008, ISBN : 978-007-126378-8, Chapter 9.
4. G. Ungerboeck, "Channel Coding with Multilevel / Phase Signals, IEEE Transactions on Information Theory Vol. IT-28, No. 1, January 1982.
5. My own lecture notes.