Structured Sequences

In addition to information symbols, *structured sequences* contain redundant (extra) symbols that if chosen with purpose can improve the reliability of information transmission through a noisy channel. These redundant symbols allow for the detection and in many cases the correction of erroneous symbols in the received symbol stream.

Human languages are structured sequences with a great deal of redundancy. Claude Shannon estimated the redundancy in the English language to be at least 50% – half the letters in our words are not needed to convey meaning! Word games like crossword puzzles, cryptograms, “Hangman” and “Wheel of Fortune” would not be possible or any fun without redundancy.

Parity Check

From a coding perspective the simplest structured sequences are ones where a single parity check symbol is added to a sequence of k binary information symbols. This parity symbol is chosen to make the number of “ones” in the total sequence including the parity symbol either even (*even parity*) or odd (*odd parity*). The block length is $n = k + 1$ and the code rate is $r = k/(k + 1)$.

At the receiver the number of “ones” in a received message block is computed. If even in the case of an even parity scheme either no errors occurred in transmission or an even number occurred which is undetectable. If the computed parity is odd then it is certain that an odd number of errors occurred.

Parity check schemes are used in conjunction with half-duplex or full-duplex two-way transmission systems to detect and then request the retransmission of erroneous data. Such systems are known as *automatic retransmission query* or ARQ systems.

Forward Error Correction

In many cases ARQ systems are not feasible to implement:

- A reverse transmission channel is not available or too expensive
- The delay associated with retransmission of erroneous data is too long
- The expected number of errors requires too many retransmissions

In such cases additional coding and system complexity that permits not only the detection of errors but the correction of some or all of these errors is merited. These systems are called *forward error correction* or FEC systems and are the focus of the rest of this course.

An excellent example of a FEC system is the NASA/JPL Deep Space Network used to communicate with the Voyager satellites. Voyager I is now 21 billion kilometers from Earth. The round-trip light time is 1.63 days and the channel is quite noisy. In these circumstances an ARQ system would simply be unworkable. Instead a very sophisticated FEC system using concatenated Reed-Solomon and convolutional codes is used.

Block Codes

For a *block code* there are k information symbols for every block of n symbols ($n > k$) in the data stream. This means there are $m = n - k$ redundant or parity symbols in each block. The code rate is $r = k/n$ and the corresponding bandwidth expansion is $1/r$.

Block codes may be thought of as a one-to-one mapping from 2^k possible k-bit message words to 2^k n-bit code words. This mapping may be accomplished via a look-up table or by a rule. The k-bit message words are also called k-tuples and the n-bit code words are called n-tuples.

The larger n is relative to k the lower the code rate and the higher the bandwidth expansion. On the other hand larger values of n relative to k make it possible to space code words farther apart allowing for increased error detection and correction. This is a fundamental trade-off in coding.

Vector Spaces

The set of all binary n-tuples $\boldsymbol{V_n}$ is called a *vector space* over the binary (finite) field of two elements $\{0, 1\}$. The binary field has two operations, addition and multiplication, defined such that the result of all operations remain within the same set of elements $\{0, 1\}$. This property is known as *closure*. Binary field addition and multiplication, also known as *modulo-2 arithmetic*, are defined as follows:

+	0	1
0	0	1
1	1	0

•	0	1
0	0	0
1	0	1

Addition and multiplication are commutative in a binary field. Further, multiplication is distributive over addition. As a note of interest, subtraction is identical to addition in a binary field.

Binary field addition (and subtraction) can be implemented in hardware with logical *exclusive-or* (XOR) gates while multiplication can be implemented with logical *and* (AND) gates.

Vector Subspaces

A subset $\boldsymbol{S}$ of the vector space $\boldsymbol{V_n}$ is called a *subspace* if the following two conditions hold:

- The all-zero vector is in $\boldsymbol{S}$.
- The sum of any two vectors in $\boldsymbol{S}$ is also in $\boldsymbol{S}$ (closure).

Linear Block Codes

Suppose $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ are two code words (or equivalently two code vectors) in a (n, k) binary block code. This block code is a *linear block code* if and only if the vector $\boldsymbol{c_1} + \boldsymbol{c_2}$ is also a code vector where addition is over the binary field. A linear block code is then one in which vectors outside the subspace defined by the code vectors cannot be created by the addition of legitimate code vectors within the subspace.

Example:

Suppose we have the following subset $\boldsymbol{C}$ of $\boldsymbol{V_4}$:

0000 0101 1010 1111

It is easy to verify that $\boldsymbol{C}$ is a subspace of $\boldsymbol{V_4}$ therefore the code words represented by code vectors in $\boldsymbol{C}$ are a linear block code.

This is an important practical result. For a linear block code any of the 2^k n-tuple code words $\boldsymbol{c}$ can be generated from a basis set of k linearly independent n-tuples:

$$\boldsymbol{c} = m_1\boldsymbol{g_1} + m_2\boldsymbol{g_2} + \cdots + m_k\boldsymbol{g_k}$$

where the m_i are message word digits and addition and multiplication are over the binary field.

Define a *generator matrix*:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_k \end{bmatrix}$$

Using matrix multiplication *defined over the binary field*, the n-tuple row code vector $\boldsymbol{c}$ in terms of the k-tuple row message vector $\boldsymbol{m}$ and the $k \times n$ generator matrix $\boldsymbol{G}$ is:

$$\boldsymbol{c} = \boldsymbol{mG}$$

For all but the shortest block codes a table lookup encoding scheme is impractical. If the block code is linear efficient matrix multiplication can be used instead. The Golay(24,12) code has $2^{12} = 4096$ valid 24-symbol code words. Code words can be generated using a 12×24 generator matrix and binary field addition and multiplication.

Systematic Codes

There is no requirement that the k information symbols be contiguous within a block. If they are the code is said to be *systematic*. If they are not the code is *nonsystematic*. The generator matrix for a systematic code will have an easily identifiable identity matrix within it.

Often a systematic code is structured so the k information symbols are at the beginning of a code word and the m parity symbols are at the end. In this case the generator matrix takes the form:

$$\boldsymbol{G} = [\boldsymbol{I} \quad \boldsymbol{P}]$$

where $\boldsymbol{I}$ is a $k \times k$ identity matrix and $\boldsymbol{P}$ is a $k \times m$ *parity array matrix*.

If the rules (equations) for determining the various parity symbols p_j from the message symbols m_i are known then the parity array matrix $\boldsymbol{P}$ can easily be created – the entries of the jth column of $\boldsymbol{P}$ are the coefficients of the linear combination of the m_i that define p_j.

Examples:

The two parity symbols of the Repetition(3,1) code are defined by:

$$p_1 = m_1$$

$$p_2 = m_1$$

The systematic generator matrix is:

$$\boldsymbol{G} = [1 \quad 1 \quad 1]$$

The single parity symbol for the even Parity Check(8,7) code discussed above is found by:

$$p_1 = m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7$$

The systematic generator matrix is then:

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

Hamming Distance and Weight

We need a metric for the distance between two code words or equivalently the distance between the two code vectors representing those code words. A useful one is the *Hamming distance* defined as the number of positions where the two code words or their corresponding code vectors differ.

The *Hamming weight* is defined as the number of "ones" in a code word or equivalently its distance from an all-zero code word.

Since we are operating in a binary field the Hamming weight of the sum of two code vectors (or words) is equal to the Hamming distance between the two code vectors (or words).

Example:

Suppose we have two binary code words:

$$\boldsymbol{A} = [1\ 0\ 0\ 1\ 0\ 1\ 1\ 0]$$

$$\boldsymbol{B} = [0\ 1\ 1\ 1\ 1\ 0\ 1\ 0]$$

The Hamming distance between them is:

$$d(\boldsymbol{A}, \boldsymbol{B}) = 5$$

The Hamming weights are:

$$w(\boldsymbol{A}) = 4$$

$$w(\boldsymbol{B}) = 5$$

Note:

$$\boldsymbol{A} + \boldsymbol{B} = [1\ 1\ 1\ 0\ 1\ 1\ 0\ 0]$$

$$w(\boldsymbol{A} + \boldsymbol{B}) = 5 = d(\boldsymbol{A}, \boldsymbol{B})$$

Using MATLAB

MATLAB has a number of useful functions for performing the various calculations associated with linear block codes.

Define the two code vectors ***A*** and ***B*** from the example above:

```
>> A = [1 0 0 1 0 1 1 0]

A =

     1     0     0     1     0     1     1     0

>> B = [0 1 1 1 1 0 1 0]

B =

     0     1     1     1     1     0     1     0
```

The weights of ***A*** and ***B*** are found by:

```
>> weightA = sum(A)

weightA =

     4

>> weightB = sum(B)

weightB =

     5
```

The distance between ***A*** and ***B*** is given by:

```
>> distanceAB = sum(A ~= B)

distanceAB =

     5
```

The sum of $\boldsymbol{A}$ and $\boldsymbol{B}$ is:

```
>> sumAB = mod(A+B, 2)

sumAB =

     1     1     1     0     1     1     0     0
```

The weight of $\boldsymbol{A}+\boldsymbol{B}$ is:

```
>> weightAB = sum(mod(A+B, 2))

weightAB =

     5
```

As expected the distance between $\boldsymbol{A}$ and $\boldsymbol{B}$ is equal to the weight of $\boldsymbol{A}+\boldsymbol{B}$.

Suppose we have a (6,3) code where:

$$p_1 = m_1 + m_3$$

$$p_2 = m_1 + m_2$$

$$p_3 = m_2 + m_3$$

The systematic generator matrix $\boldsymbol{G}$ is:

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

If the message word is $\boldsymbol{m} = [1\ 0\ 1]$ then the code word $\boldsymbol{c}$ is:

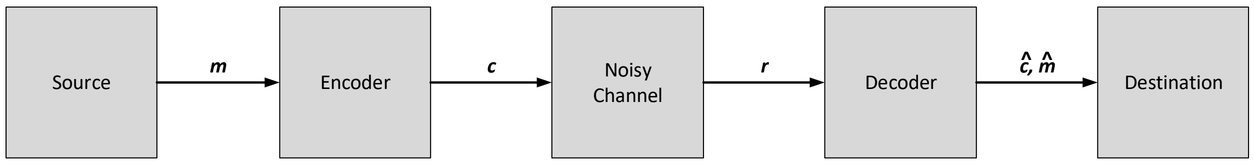
$$\boldsymbol{c} = \boldsymbol{mG} = [1\ 0\ 1\ 0\ 1\ 1]$$

Using MATLAB:

```
>> G = [1 0 0 1 1 0; 0 1 0 0 1 1; 0 0 1 1 0 1]

G =

     1     0     0     1     1     0
     0     1     0     0     1     1
     0     0     1     1     0     1


>> m = [1 0 1]

m =

     1     0     1


>> c = mod(m*G, 2)

c =

     1     0     1     0     1     1
```

Maximum Likelihood Decoding

Consider the coding system with a noisy discrete memoryless channel shown:



The decoder must produce an estimate $\hat{\boldsymbol{m}}$ of the original source message $\boldsymbol{m}$ based on the received sequence $\boldsymbol{r}$. Since there is a one-to-one correspondence between source message sequences $\boldsymbol{m}$ and code words $\boldsymbol{c}$ this is equivalent to producing an estimate $\hat{\boldsymbol{c}}$ of the code word $\boldsymbol{c}$.

A *decoding rule* is a strategy for choosing an estimated code word $\hat{\boldsymbol{c}}$ for each possible received sequence $\boldsymbol{r}$. A decoding error occurs if $\hat{\boldsymbol{c}} \neq \boldsymbol{c}$. Given that $\boldsymbol{r}$ is received the conditional error probability of a decoder error is defined as:

$$P(E|\boldsymbol{r}) = P(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{r})$$

The error probability of the decoder is then:

$$P(E) = \sum_{\boldsymbol{r}} P(E|\boldsymbol{r})\, P(\boldsymbol{r})$$

where $P(\boldsymbol{r})$ is the probability of the received sequence $\boldsymbol{r}$. $P(\boldsymbol{r})$ is independent of the decoding rule used since $\boldsymbol{r}$ is produced prior to the decoder.

An optimum decoding rule minimizes $P(E)$ or equivalently minimizes $P(E|\boldsymbol{r}) = P(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{r})$ for all $\boldsymbol{r}$. This is the same as maximizing $P(\hat{\boldsymbol{c}} = \boldsymbol{c}|\boldsymbol{r})$ for all $\boldsymbol{r}$. Using Bayes' theorem, $P(E|\boldsymbol{r})$ is minimized for a given $\boldsymbol{r}$ by choosing $\hat{\boldsymbol{c}}$ as the code word $\boldsymbol{c}$ that maximizes:

$$P(\boldsymbol{c}|\boldsymbol{r}) = \frac{P(\boldsymbol{r}|\boldsymbol{c})P(\boldsymbol{c})}{P(\boldsymbol{r})}$$

If all the source message sequences $\boldsymbol{m}$ and hence all the code words $\boldsymbol{c}$ are equally likely then maximizing $P(\boldsymbol{c}|\boldsymbol{r})$ is the same as maximizing $P(\boldsymbol{r}|\boldsymbol{c})$ because the ratio $P(\boldsymbol{c})/P(\boldsymbol{r})$ is a constant. For a discrete memoryless channel and a block code of length n:

$$P(\boldsymbol{r}|\boldsymbol{c}) = \prod_{i=1}^{n} P(\boldsymbol{r}_i|\boldsymbol{c}_i)$$

since each received symbol depends only on the corresponding transmitted symbol.

A decoder that chooses its estimate based on maximizing the expression above is called a *maximum likelihood decoder* (MLD). Because $\log x$ is a monotonically increasing function of x, maximizing $P(\boldsymbol{r}|\boldsymbol{c})$ is equivalent to maximizing the *log-likelihood function*:

$$\log P(\boldsymbol{r}|\boldsymbol{c}) = \sum_{i=1}^{n} \log P(\boldsymbol{r_i}|\boldsymbol{c_i})$$

The logarithm base is not important and can be base 10 for convenience. For a BSC at a position i in the symbol sequence, $r_i \neq c_i$ with probability p and $r_i = c_i$ with probability $1 - p$. The number of positions in the symbol sequence where $r_i \neq c_i$ is precisely equal to the Hamming distance $d(\boldsymbol{r}, \boldsymbol{c})$:

$$\log P(\boldsymbol{r}|\boldsymbol{c}) = d(\boldsymbol{r}, \boldsymbol{c}) \log p + [n - d(\boldsymbol{r}, \boldsymbol{c})] \log(1 - p)$$

$$= d(\boldsymbol{r}, \boldsymbol{c}) \log \frac{p}{1 - p} + n \log(1 - p)$$

Now $\log \frac{p}{1-p} < 0$ for $p < 0.5$ and $n \log(1 - p)$ is a constant for all $\boldsymbol{c}$.

The MLD decoding rule for a BSC chooses as its estimate $\hat{\boldsymbol{c}}$ the code word $\boldsymbol{c}$ that minimizes the distance $d(\boldsymbol{r}, \boldsymbol{c})$ between $\boldsymbol{r}$ and $\boldsymbol{c}$; that is, it chooses the code word that differs from the received sequence in the fewest number of positions. For this reason a MLD for a BSC is also referred to as a *minimum distance decoder*. The destination message estimate $\hat{\boldsymbol{m}}$ is then the message that corresponds to $\hat{\boldsymbol{c}}$.

These results can be extended to a binary AWGN channel as well by using Gaussian probability density functions for the $P(\boldsymbol{r_i}|\boldsymbol{c_i})$.

Syndrome Decoding

Conceptually maximum likelihood decoding can be accomplished with a lookup table. Each possible received n-tuple is paired with the closest valid code word. In practice this quickly becomes unwieldy. Consider again the Golay(24,12) code. There are $2^{24} = 16{,}777{,}216$ possible received sequences which would need to be mapped to the $2^{12} = 4096$ valid code words. Memory requirements and search times are unfeasible.

For linear block codes more efficient methods exist based on *syndrome decoding*. These methods make use of *error pattern* lookup tables with only $2^m = 2^{n-k}$ entries.

A *parity check matrix* is used to facilitate syndrome decoding by checking the parity symbols of the received message. For every $k \times n$ generator matrix $\boldsymbol{G}$ there is a $m \times n = (n-k) \times n$ parity check matrix $\boldsymbol{H}$. The rows of the parity check matrix $\boldsymbol{H}$ are chosen to be orthogonal to the rows of the generator matrix $\boldsymbol{G}$. In the case $\boldsymbol{G}$ is the generator matrix of a systematic code with parity array $\boldsymbol{P}$ then:

$$\boldsymbol{G} = [\boldsymbol{I} \quad \boldsymbol{P}]$$

$$\boldsymbol{H} = [\boldsymbol{P^T} \quad \boldsymbol{I}]$$

$$\boldsymbol{H}^T = \begin{bmatrix} \boldsymbol{P} \\ \\ \boldsymbol{I} \end{bmatrix}$$

In any case the rows of $\boldsymbol{G}$ and $\boldsymbol{H}$ are orthogonal and it is always true that:

$$\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$$

Since any valid code word $\boldsymbol{c}$ is a linear combination of the basis vectors that constitute $\boldsymbol{G}$:

$$\boldsymbol{c}\boldsymbol{H}^T = \boldsymbol{m}\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$$

Suppose $\boldsymbol{r}$ is the received message word when $\boldsymbol{c}$ is sent through a noisy channel. The noise can cause $\boldsymbol{r}$ to differ from $\boldsymbol{c}$ in some number of positions and we can write:

$$\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$$

The vector $\boldsymbol{e}$ represents the *error pattern* introduced by the channel. It can take on any of 2^n possible values including the all-zero vector meaning no error occurred in transmission.

The *syndrome* of $\boldsymbol{r}$ is defined as:

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{r}\boldsymbol{H}^T \\ &= (\boldsymbol{c}+\boldsymbol{e})\boldsymbol{H}^T \\ &= \boldsymbol{c}\boldsymbol{H}^T + \boldsymbol{e}\boldsymbol{H}^T \\ &= \boldsymbol{e}\boldsymbol{H}^T \end{aligned}$$

The syndrome is a parity check on the received message word $\boldsymbol{r}$ to determine whether $\boldsymbol{r}$ is a valid code word. If $\boldsymbol{r}$ is a valid code word then the syndrome $\boldsymbol{S}$ will be the all-zero vector. Note that it is possible at the same time for $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ to be a valid code word with a zero syndrome and for the error pattern $\boldsymbol{e}$ to be nonzero. This happens when the error pattern is itself a valid code word and $\boldsymbol{e}\boldsymbol{H}^T = \boldsymbol{0}$. In this case an *undetectable error* has occurred.

If $\boldsymbol{r}$ contains detectable errors then $\boldsymbol{S}$ will have one of $2^m - 1$ nonzero values. For linear codes the mapping from possible syndromes to correctable error patterns is one-to-one. Each syndrome (like the symptom of an illness) can therefore be made to point to a unique error pattern which is subsequently used for error correction.

The *standard array* is used to organize syndrome decoding. The 2^k valid code words $\boldsymbol{c}_i$ are listed in a header row beginning with the all-zero code word $\boldsymbol{c}_0$. The next row is started with any n-tuple $\boldsymbol{e}_1$ that is not valid code word. The row is completed with the entries $\boldsymbol{c}_i + \boldsymbol{e}_1$ each placed below $\boldsymbol{c}_i$ in the header row. All entries in this row have the same syndrome $\boldsymbol{S}_1 = \boldsymbol{e}_1\boldsymbol{H}^T$. This process is repeated for each row starting with a n-tuple $\boldsymbol{e}_j$ that is not in one of the earlier rows. When completed the syndromes and the standard array will look like:

Syndrome	**Code Words**				
$\boldsymbol{e}_0$	$\boldsymbol{c}_0$	$\boldsymbol{c}_1$	$\boldsymbol{c}_2$		$\boldsymbol{c}_{2^k-1}$
$\boldsymbol{e}_1\boldsymbol{H}^T$	$\boldsymbol{e}_1$	$\boldsymbol{c}_1+\boldsymbol{e}_1$	$\boldsymbol{c}_2+\boldsymbol{e}_1$	$\cdots$	$\boldsymbol{c}_{2^k-1}+\boldsymbol{e}_1$
$\boldsymbol{e}_2\boldsymbol{H}^T$	$\boldsymbol{e}_2$	$\boldsymbol{c}_1+\boldsymbol{e}_2$	$\boldsymbol{c}_2+\boldsymbol{e}_2$	$\cdots$	$\boldsymbol{c}_{2^k-1}+\boldsymbol{e}_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{e}_j\boldsymbol{H}^T$	$\boldsymbol{e}_j$	$\boldsymbol{c}_1+\boldsymbol{e}_j$	$\boldsymbol{c}_2+\boldsymbol{e}_j$	$\cdots$	$\boldsymbol{c}_{2^k-1}+\boldsymbol{e}_j$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{e}_{2^m-1}\boldsymbol{H}^T$	$\boldsymbol{e}_{2^m-1}$	$\boldsymbol{c}_1+\boldsymbol{e}_{2^m-1}$	$\boldsymbol{c}_2+\boldsymbol{e}_{2^m-1}$	$\cdots$	$\boldsymbol{c}_{2^k-1}+\boldsymbol{e}_{2^m-1}$

The standard array has $2^m = 2^{n-k}$ rows including the header row of valid code words. There are 2^k columns and a total of 2^n entries representing all the possible received message n-tuples. Each row is called a *coset* as all the entries in that row share the same syndrome. The first entry in each row is called the *coset leader*. The coset leader is the correctable error pattern that is applied to a received message word found in that row to form an estimate of the code word sent.

When constructing the standard array the choice of the coset leader for a given row is not unique. For a BSC an error pattern of smaller weight is more probable than one of larger weight. For maximum likelihood (minimum distance) decoding each coset leader should therefore be chosen in turn as a vector of least weight from the remaining available vectors.

Once a standard array for maximum likelihood decoding has been constructed a lookup table of the 2^m syndromes and coset leaders (correctable error patterns) is formed for the actual decoding process. Decoding a received message vector $\boldsymbol{r}$ consists of three steps:

1. Compute the syndrome $\boldsymbol{S} = \boldsymbol{r}\boldsymbol{H}^T$
2. Locate the corresponding coset leader $\boldsymbol{e}$
3. Decode the received message vector as $\hat{\boldsymbol{c}} = \boldsymbol{r} + \boldsymbol{e}$

If $\hat{\boldsymbol{c}} = \boldsymbol{c}$ the received message word has been correctly decoded. If however $\hat{\boldsymbol{c}} \neq \boldsymbol{c}$ then a *decoding error* has occurred.

As m increases this table lookup scheme becomes impractical. For many useful codes the lookup tables are replaced by algorithms. These algorithms (which can be quite complicated) depend on additional properties of the codes beyond a linear structure and will be discussed later.

Minimum Distance of a Linear Block Code

Consider the set of the distances between all pairs of code words for a given linear block code. The smallest member of this set is the *minimum distance* of the code denoted by d_{min}. The minimum distance of a linear code characterizes its ability to detect and correct errors – the greater the minimum distance, the stronger the code.

Rather than calculate d_{min} by directly comparing all the pairwise distances of a code we note that the distance between two code words is equal to the weight of the sum of the two code words. Since all code words of a linear code can be expressed as the sum of two other code words, the task of determining the minimum pairwise distance simplifies to finding the nonzero code word with the minimum weight.

$$d_{min}(\boldsymbol{C}) = \min_{all\ \boldsymbol{c_i} \neq \boldsymbol{0}} w(\boldsymbol{c_i})$$

Error Detection and Correction Limits

The *error-correcting capability* t of a code is defined as the maximum number of guaranteed correctable errors per code word and is given by:

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor$$

where $\lfloor x \rfloor$ is the largest integer not to exceed x also known as the *floor* function.

Often a code that corrects all sequences with t or fewer errors can also correct certain sequences with $t + 1$ errors. This can happen because a (n, k) linear code is capable of correcting a total of $2^m = 2^{n-k}$ error patterns.

The *error-detecting capability* e of a code is defined as the maximum number of guaranteed detectable errors per code word and is given by:

$$e = d_{min} - 1$$

A code can be used to simultaneously correct α errors and detect β errors where $\beta \geq \alpha$ provided:

$$\alpha + \beta \leq d_{min} - 1$$

For example the Golay(24, 12) code has a minimum distance of 8. It can therefore detect all sequences with 4 errors and correct all sequences with 3 errors.

Error Probability

If a t-error correcting code is used strictly for error correction on a BSC with transition probability p then the probability of a message error is upper bounded by:

$$P_M \leq \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

where $\binom{n}{j} = \frac{n!}{j!(n-j)!}$ is the binomial coefficient. It is the number of possible combinations of n items taken j at a time without regard to order also referred to as n *choose* j.

If the decoder fixes all sequences with t or fewer error and no sequences with $t+1$ errors then the upper bound becomes an equality. Such decoders are called *bounded distance decoders* and:

$$P_M = \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

The probability of a bit error is then approximated by:

$$P_B \approx \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j}$$

For a binary AWGN channel the transition probability is related to the coded channel symbol-to-noise ratio by:

$$p = \frac{1}{2} \operatorname{erfc} \sqrt{\frac{E_s}{N_0}}$$

The coded channel symbol-to-noise ratio is related to the uncoded source message bit-to-noise ratio by:

$$\frac{E_s}{N_0} = r \frac{E_b}{N_0}$$

where r is the code rate.

Characteristics of Codes

As discussed earlier the requirement that channel coded information arrive at the destination at the same rate as uncoded information causes the coded symbol-to-noise ratio E_s/N_0 to be less than the uncoded bit-to-noise ratio E_b/N_0 by a factor of r given constant signal power. This will necessarily cause the channel coded symbol error rate to be worse than the uncoded rate. To be useful a code needs to more than overcome this degradation in noise ratios. This characteristic is shown on the graph on the next page by the arrow labeled "A".

Good codes also allow the error rate to be maintained at a given level with less signal power than that needed for uncoded messaging. This is shown on the graph by the arrow labeled "B". We define the *coding gain* of a code in decibels at a specified error rate as the reduction in signal power allowed by the use of the code:

$$Coding\ Gain_{(\mathrm{dB})} = \frac{E_b}{N_0}_{uncoded\ (\mathrm{dB})} - \frac{E_b}{N_0}_{coded\ (\mathrm{dB})}$$

As the bit-to-noise ratio degrades all codes have a threshold or crossover point where they perform worse than uncoded messaging. This is point "C" on the graph. Below the crossover point the error correcting capability of the code has been exceeded – there are more errors in a block than the code can correct. The parity bits consume energy but give nothing back of benefit in return.

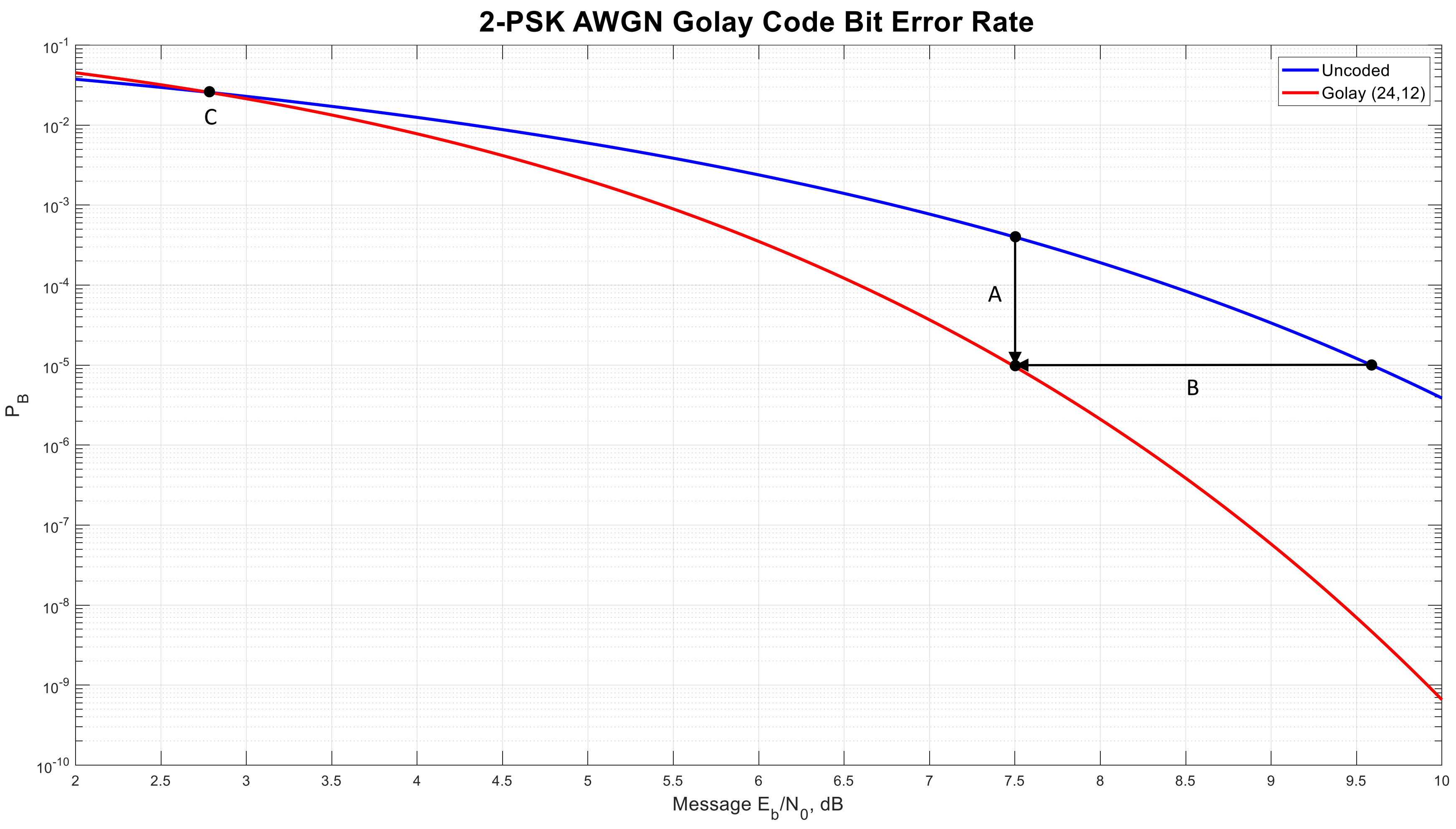
Hamming Bound

The *Hamming bound* gives an important limitation on the efficiency with which an error-correcting code can utilize the space in which its code words are embedded. This limit is also known as the *sphere-packing bound* or the *volume bound*. If a binary code has $M = 2^k$ code words of block length n and can correct t errors then:

$$\sum_{j=0}^{t} \binom{n}{j} \leq 2^{n-k}$$

Perfect Codes

A t-error correcting code is called a *perfect code* if its standard array has all the error patterns of t and fewer errors and no others as coset leaders. Equivalently, a perfect code attains the Hamming bound. The single-error correcting Hamming codes and the triple-error correcting Golay(23,12) code are the only known nontrivial perfect binary codes.


2-PSK AWGN Golay Code Bit Error Rate
Uncoded
Golay (24,12)
A
B
C
P_B
Message E_b/N_0, dB
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10

Example:

Determine the probability of a bit error and the coding gain for the Golay(24,12) code given an uncoded source message bit-to-noise ratio of 7.5 dB.

The Golay(24,12) code corrects all sequences with three or fewer errors so $t = 3$.

Proceeding:

$$\frac{E_b}{N_0} = 10^{0.75} = 5.6234$$

$$\frac{E_s}{N_0} = \frac{12}{24} \cdot 5.6234 = 2.8117$$

$$p = \frac{1}{2}\mathrm{erfc}\sqrt{2.8117} = 0.008861$$

This sets all the parameters for the coded bit error rate (BER) calculation:

$$P_B = \frac{1}{24}\sum_{j=4}^{24} j \binom{24}{j} \cdot 0.008861^{j} \cdot 0.9911^{24-j}$$

Using MATLAB for this summation:

$$P_B = 9.561 \times 10^{-6} \approx 10^{-5}$$

From the graph above the uncoded bit error rate is about 4×10^{-4}. Using the Golay(24,12) code at this signal power therefore results in a 40*x* improvement in bit error rate compared to uncoded.

Recall the uncoded BER for $\frac{E_b}{N_0} = 9.6$ dB is also about 10^{-5}.

The coding gain for the Golay(24,12) code at a BER of 10^{-5} is then:

$$\text{Coding Gain} = 9.6 - 7.5 = 2.1\ \text{dB}$$

This means we can achieve the same 10^{-5} uncoded bit error rate using the Golay(24,12) code but with 2.1 dB less power. This is a 38% reduction in required signal power.