Practical 1

REII 411

by:

S. Jacholke	23671750
R.J. von Wielligh	23517972

submitted in persuit of the degree

BACHELOR OF ENGINEERING
in
COMPUTER AND ELECTRONIC ENGINEERING

Lecturer: M. Feirrera

North-West University Potchefstroom Campus

January 2015

It all starts here

NORTH-WEST UNIVERSITY ®
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
POTCHEFSTROOM CAMPUS

CONTENTS

1 INTRODUCTION 1
2 BACKGROUND 2
2.1 FEC Concept 2
2.2 Hamming distance 5
2.3 Hamming code 5
2.4 Hamming (7,4) details 6
2.4.1 Table generation 7
2.4.2 Matrix generation 7
2.4.3 Error probability 8
3 METHODOLOGY 10
4 RESULTS 13
5 DISCUSSION 15
6 CONCLUSION 16
A APPENDIX - SOURCE 17

LIST OF FIGURES

Figure 1 Venn diagram indicating error correction for 4 bits 4
Figure 2 Flow diagram to obtain bit error rate 11
Figure 3 Plot of Bit Error Rate (BER) over Signal to Noise Ratio (SNR) 14
Figure 4 Picture plot of hamming distance matrix 14

LIST OF TABLES

Table 1 Error indication 3
Table 2 Even parity bit verification 6
Table 3 Transmitted bits 7
Table 4 Hamming Distance Matrix 13

ACRONYMS

AWGN	Additive White Noise
FEC	Forward Error Correction
SNR	Signal to Noise Ratio
BER	Bit Error Rate
CRC	Cyclic Redundency Check

INTRODUCTION

By adding extra data as redundancy to an existing message on can check for errors. Error detection is used widely in a variety of applications. On the internet it is typically used in Ethernet frames. Frames with incorrect checksums are discarded.

Errors may also be corrected. Common methods make use of extra bits to locate the location of the errors. Common DRAM memory uses error correcting code as protection against soft errors.

In this practical we will focus on forward error correction and in particular investigate the Hamming 7,4 code.

Forward Error correction allows for controlling errors in data transmissions over lossy communication channels.

We will first investigate the use of error correction and then design a simplistic system that models a Additive White Noise (AWGN) channel and plot the bit error rate. The results will be documented and conclusions will be drawn about the results.

BACKGROUND

2.1 FEC CONCEPT

Forward Error Correction (FEC) is sometimes wrongly interpreted as error detection. There are three error detection methods commonly used, namely: Parity, Checksum and Cyclic Redundency Check (CRC) and is used to only detect errors. Whereas FEC is a way of adding redundancy to messages so that the receiver can both detect and correct common errors.

FEC is a technique mainly used in telecommunications, for controlling errors in data transmission over noisy and unreliable channels.

FEC has two main categories [1]:

- Block codes:
  Block codes have a fixed size, and can be seen as a packet of bits or symbols. Block codes can also be hard-decoded in polynomial time depending on the block length.
- Convolutional Codes:
  Convolutional codes works on a bit or symbols streams of arbitrary length. The most common soft decoding used is the Viterbi algorithm, because it offers asymptotically optimal decoding efficiency with increasing constraint length of the convolutional code.

FEC works by adding redundancy to information being transmitted by means of an algorithm. The redundant bits can consist of many of the original information bits and this original information could appear in the encoded output. If the original information appears in the encoded output it the code is known as systematic otherwise it is called as non-systematic [2].

The easiest example of FEC is when each data bit gets transmitted three times, which in practice is called a repetition code. Table 1 illustrates the eight different versions of the output, which the receiver might see.

Using error correction codes may not be suitable in every application, however there are advantages in doing so:

Advantages:

- Sometimes a no-feedback channel is necessary
- Long delay path
- One-way Transmission

Table 1: Error indication

Triplet Received	Interpreted as
000	0 (No Error)
001	0
010	0
100	0
111	1 (No Error)
110	1
101	1
011	1

- Avoids multicast problems

Disadvantages:

- Computationally expensive
- Requires over-transmission

Figure 1 indicates the hamming error correction for 4 bits. In (a) 4 bits are assigned to the overlapping circles. In (b) a parity is assigned to all non-overlapping segments such that all bits in a line are even. In (c) the bit in the overlap between A and C is flipped. In (d) we can see that the parity bits indicates which line of bits are not erroneous. Since [100] is odd the erroneous bit must be in the overlap between A and C.

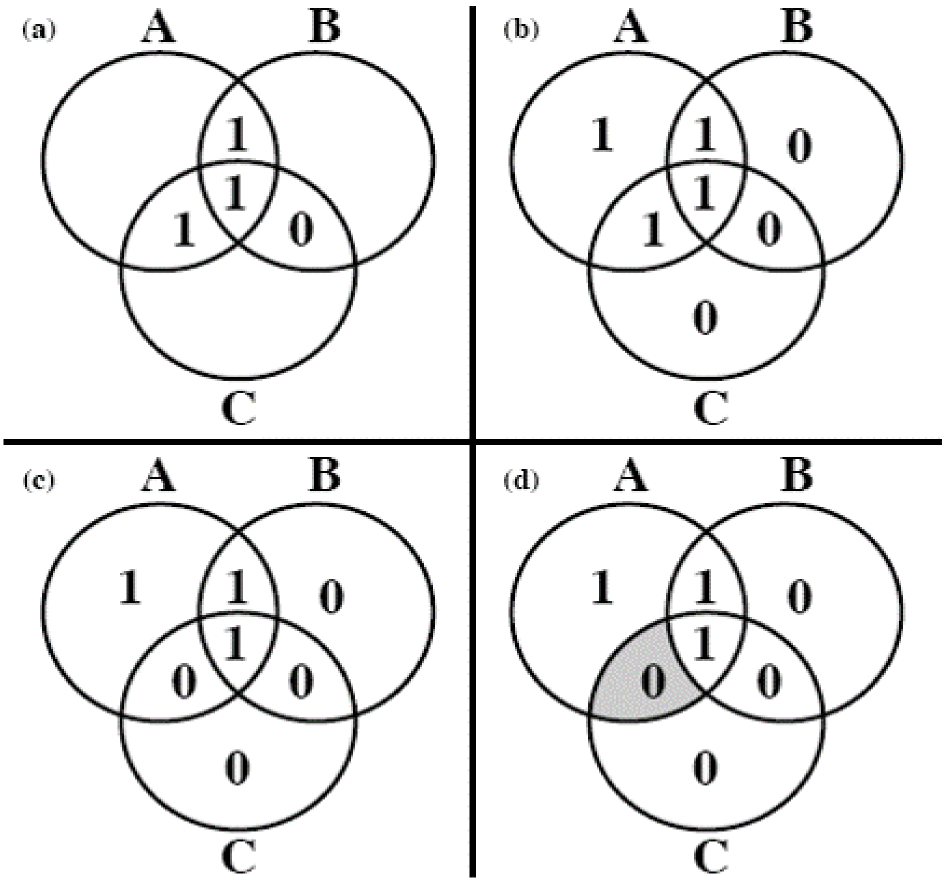

Figure 1: Venn diagram indicating error correction for 4 bits

2.2 HAMMING DISTANCE

The hamming distance is the number of bits that change from one bit pattern to another. For example, the hamming distance between: [1000101] and [1010001] is two.

Hamming distance may indicate the error correction capabilities of a block code. For a minimum hamming distance of d:

- It is possible to detect any number of errors less than d.
- It is possible to correct any number of errors less than $\frac{d}{2}$. [1]

In addition if we want to detect at least n errors then the minimum hamming distance d must be at least [3]:

$$d \geqslant n + 1$$

2.3 HAMMING CODE

The Hamming code is a set of error detections codes that is used to detect and correct bit errors that occur when data is moved or stored in a computer.

Just like most error checking codes, the hamming code makes use of parity bits. These are bits added to the data, so that the data can be validated and checked when it is read or after it's been received through a data transfer of some sort. When sufficient parity bits are used, it's possible for the error detection code not just to identify the error in the data but also locate that error.

FEC and hamming code has a direct link to each other, this is the ability of the receiving station to correct errors in the received data of the data transmission and it can increase the throughput of a data link if it is very noisy. To enable this, the transmission station must add extra data to the original data being transmitted and these extra data bits are called error correction bits. Although this method is very accurate it isn't always the cheapest method to use, as sometimes it is just easier and faster to retransmit the data if it is possible. Hamming code makes FEC less expensive if it is used through the use of a block parity mechanism.

The Hamming code can be computed through parity. This involves counting the amount of ones in a set of data and adding either a zero or a one to make the count odd or even , leaving the parity odd or even, by means if a parity bit addition [4].

For binary addition we use the XOR operation, that is:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

1 if d is odd then $\frac{d-1}{2}$ errors can be corrected

$$1 + 1 = 0$$

Thus for even parity we require that the binary sum of all ones add up to zero.

Table 2: Even parity bit verification

2 Bit String	Parity Bit	Verification
00	0	0+0+0=0
01	1	0+1+1=0
10	1	1+0+1=0
11	0	1+1+0=0
x1 x2	x1 + x2	x1 + x2 + x1 + x2 = 0

If an odd parity is picked up it means that noise has flipped a bit in the data that was transmitted. In case of the hamming code, it detects two bit errors by using more than one parity bit, computed in different combinations in the data.

The number of parity bits needed can be calculated by the hamming rule:

$$d + p + 1 <= 2^p$$

where d is the amount of data bits and p the amount of parity bits

2.4 HAMMING (7,4) DETAILS

For the Hamming (7,4) code we encode four data bits with three additional parity bits. The parity is three even parity bits for four data bits. There are four possible combinations however only three are used. Let our data bits be given by

$$[d_1 d_2 d_3 d_4]$$

Then the possible parity bits are:

$$p_1 = d_2 + d_3 + d_4 \tag{1}$$

$$p_2 = d_1 + d_3 + d_4 \tag{2}$$

$$p_3 = d_1 + d_2 + d_4 \tag{3}$$

$$p_4 = d_1 + d_2 + d_3 \tag{4}$$

Two common methods are used to generate hamming codes, one is by matrix multiplication and the other is by table.

2.4.1 *Table generation*

For the table generation we generate the hamming code by:

$$\text{code} = [d_4 d_3 d_2 p_1 d_1 p_2 p_3] \tag{5}$$

The ordering of the data bits to the parity bits is important, as it is used to generate the syndrome.

When a code word is received we want retrieve the original message. Let the received code word be in the form of equation 5. We calculate new parity of the received code word by equations 1 - 3. Let pr be the parity bits of the received code word and pn be the parity bits recalculated from the data bits from the received code words. pr and pn are in the form of $[p_1 p_2 p_3]$. Then the syndrome is given by the XOR operation between pr and pn.

If the syndrome is zero then no error has been detected. If the syndrome contains a single bit set to 1, then the error occurred in a parity bit and the original message is intact. If the syndrome contains more than a single bit set to 1, then the decimal value of the syndrome indicates the bit position where the error has occurred. The error can be rectified by flipping the position.

For example:
For the message [1010] the code word is [1011010] Suppose a single bit error occurs at position 5 then the erroneous received code would be [1011110] The old parity is [110], the new parity is [011]. Thus the syndrome of [101] indicates that we need to flip position 5.

2.4.2 *Matrix generation*

Table 3: Transmitted bits

Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Using a different arrangement of parity bits:

$$p_1 = d_1 + d_2 + d_4$$

$$p_2 = d_1 + d_3 + d_4$$

$$p_3 = d_2 + d_3 + d_4$$

Consider a 4x4 identity matrix:

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Then by constructing a new matrix from I where rows 1,2 and 4 correspond to p_1, p_2 and p_3 respectively we obtain $\mathbb{G}$.

$$\mathbb{G} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The syndrome matrix $\mathbb{H}$ is a mapping of table 3. The syndrome matrix is multiplied by the received code vector. If the resulting syndrome is zero it indicates that there is no error, otherwise the syndrome indicates the position of the error.

$$\mathbb{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Let $\vec{c}$ be the resulting code word for a message $\vec{m}$. $\vec{c}$ is then obtained by:

$$\vec{c} = \mathbb{G}\vec{m}$$

To find the position of the error in $\vec{c}$:

$$\vec{e} = \mathbb{H}\vec{c}$$

The bits $\vec{e}$ modulo 2 indicates the position of the error.

For example: For $e = [121]$ the resultant bits after modulo 2 are $[101]$ which indicates that bit 5 needs to be flipped.

2.4.3 *Error probability*

The probability of a bit error of the Hamming code is smaller than the probability of a block error since a single bit can be corrected.

The probability of a block error of the Hamming code is the sum of probabilities of more than one error in the block.

$$p_B = \sum_{r=2}^{7} \binom{7}{2} f^r (1-f)^{7-r}$$

For high noise (2 bits are flipped) it may result in the Hamming decoder erroneously flipping a third bit. Thus the received code word differs in three bits from the original code. Since the probability is the same for any bit to be corrupted the probability of a single bit error is:

$$p_b \approx \frac{3}{7} p_B$$

METHODOLOGY

A system will be designed that implements forward error correction by means of the hamming code.

The goal is to transmit four bits of information over a network that implements FEC we will design the system to send each possible arrangement of 4 bits over the network.

Since only groups of 4 bits will be sent over the network the Hamming (7,4) block code will be used. Refer to section 2.4 for an overview. In this section we will discuss the implementation and design behind the system.

In order to utilize forward error correction an error must first be induced into the sent codes. Errors will be induced by utilizing the awgn function in Matlab. awgn adds white Gaussian noise to a signal for a given SNR. The awgn function returns a float pattern and not a bit pattern, thus we clamp the values to the closest bit, either a 1 or a 0.

In order to obtain the bit error rate we proceed as follows:

1. Generate a 4 bit message
2. Generate the Hamming(7,4) code of the message
3. Add noise to the code for a given SNR
4. Correct the code using Hamming(7,4) error correction
5. Decode the message
6. Compare the errors to the original message

See figure 2 for a flow diagram illustrating the process. The BER wil be plotted over several SNR(db) values in order to obtain a graph showing the BER over SNR. The same process will also be repeated for a message that is not sent using FEC. Both processes will be plotted on the same graph for comparison.

Since the BER will be based on a random sample due to the randomness of the white Gaussian noise, a large number of iterations for the BER at each SNR level will be done and the average will plotted. This is also known as the Monte Carlo method.

The minimum Hamming distance for a error correction is three. In order to obtain the Hamming distance we XOR two byes and add the amount of 1s. In Matlab, we do this for two bit patterns bits1,bits2 by:

```
        sum(bits1 ~= bits2)
```

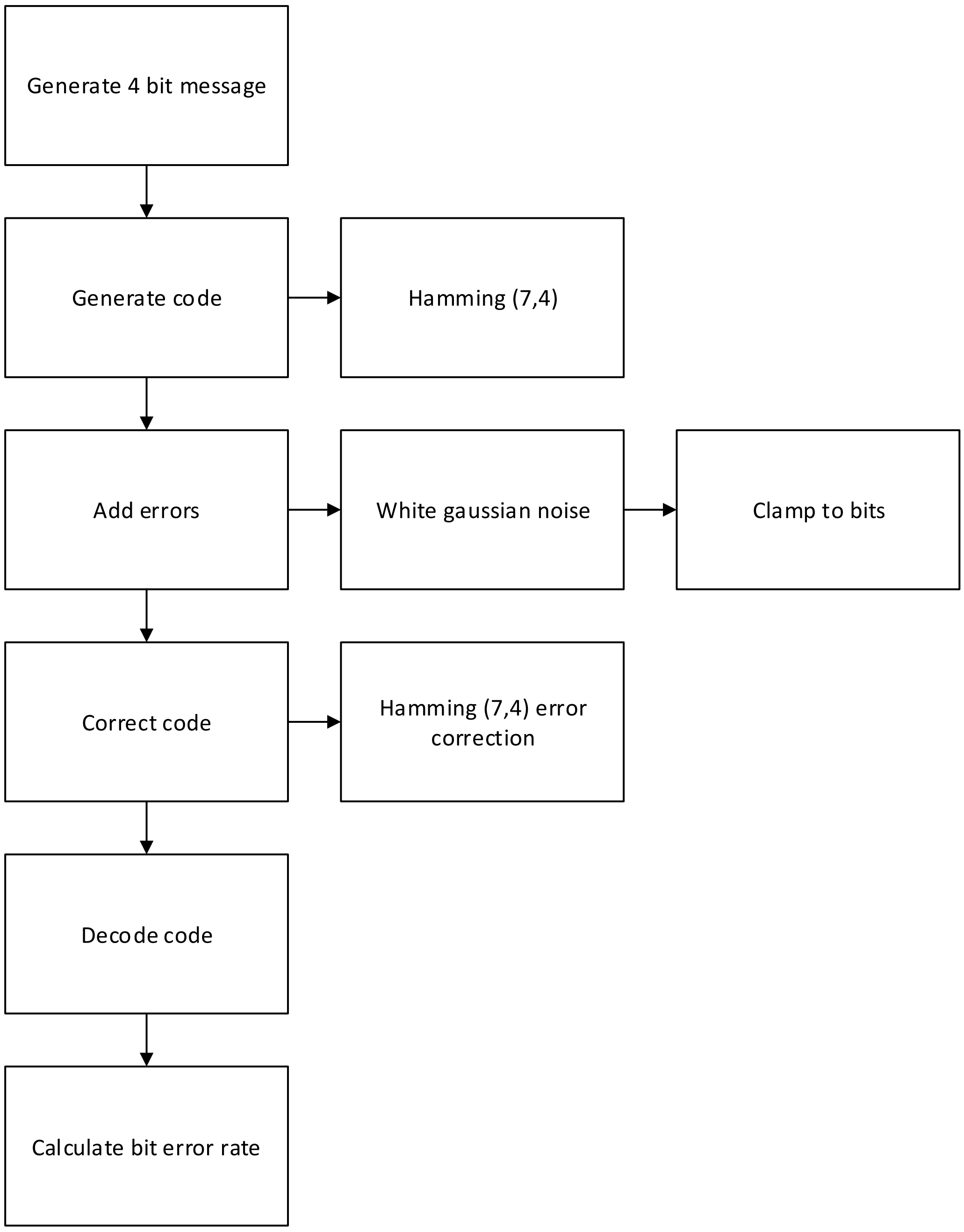

Figure 2: Flow diagram to obtain bit error rate

The code words for all 16 combinations of a 4 bit message will be generated and the minimum Hamming distance will be obtained.

In order to obtain the results a script is written in Matlab. See the appendix A for the full source code. The communications toolbox of Matlab is used in order to avoid reimplementing the Hamming Code encoding and decoding.

The communication channel is assumed to be rather simple and is assumed to have AWGN for a given SNR.

As stated previously the AWGN can vary greatly from one iteration to another. Thus in order to obtain a more accurate simulation the result is carried over a large number of iterations and the mean bit error rate is used. This also ensures that results are repeatable. The accuracy of the simulation can be increased by increasing the amount of trials. However onwards of 5000 iterations the trade off for accuracy is not worth the time.

Using AWGN does not model a more complex communication medium, however it affords simplicity of design while still allowing fairly accurate predictions. Furthermore it may be applied to predict the error rate of communications that may follow a similar noise pattern.

The use of white Gaussian noise also allows us to make the assumption that the communication channel is symmetric with respect to bit errors. Each bit has the same probability of being flipped. This assumption was used in section 2.4.3

RESULTS

Figure 3 shows the bit error rate using no correction and with hamming correction. The result was obtained using 50 000 iterations.

The minimum hamming distance obtained was 3 and the maximum 7.

```
script.m:
The minimum hamming distance between code words is: 3
```

A 16x16 matrix was populated with the hamming distance of the code words with each other from the data. In table 4 the main diagonal represents the hamming distance of a code word with respect to itself (thus 0), and is thus not considered the minimum hamming distance

The data from table 4 is also visually represented in figure 4. Blocks in teal represents the hamming distance of 3, yellow indicates a hamming distance of 4 and maroon indicates a hamming distance of 7.

Table 4: Hamming Distance Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	3	4	3	3	4	3	4	3	4	3	4	4	3	4	7
2	3	0	3	4	4	3	4	3	4	3	4	3	3	4	7	4
3	4	3	0	3	3	4	3	4	3	4	3	4	4	7	4	3
4	3	4	3	0	4	3	4	3	4	3	4	3	7	4	3	4
5	3	4	3	4	0	3	4	3	4	3	4	7	3	4	3	4
6	4	3	4	3	3	0	3	4	3	4	7	4	4	3	4	3
7	3	4	3	4	4	3	0	3	4	7	4	3	3	4	3	4
8	4	3	4	3	3	4	3	0	7	4	3	4	4	3	4	3
9	3	4	3	4	4	3	4	7	0	3	4	3	3	4	3	4
10	4	3	4	3	3	4	7	4	3	0	3	4	4	3	4	3
11	3	4	3	4	4	7	4	3	4	3	0	3	3	4	3	4
12	4	3	4	3	7	4	3	4	3	4	3	0	4	3	4	3
13	4	3	4	7	3	4	3	4	3	4	3	4	0	3	4	3
14	3	4	7	4	4	3	4	3	4	3	4	3	3	0	3	4
15	4	7	4	3	3	4	3	4	3	4	3	4	4	3	0	3
16	7	4	3	4	4	3	4	3	4	3	4	3	3	4	3	0

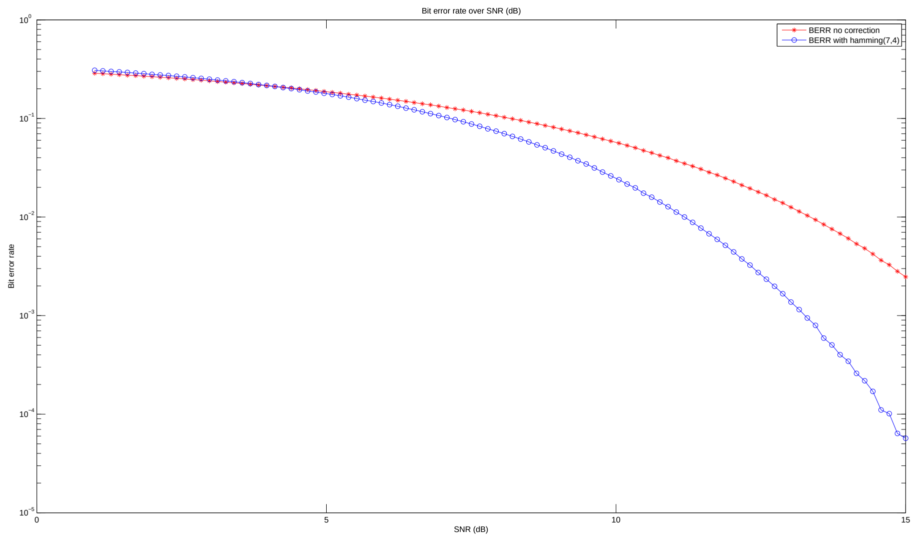

Figure 3: Plot of BER over SNR

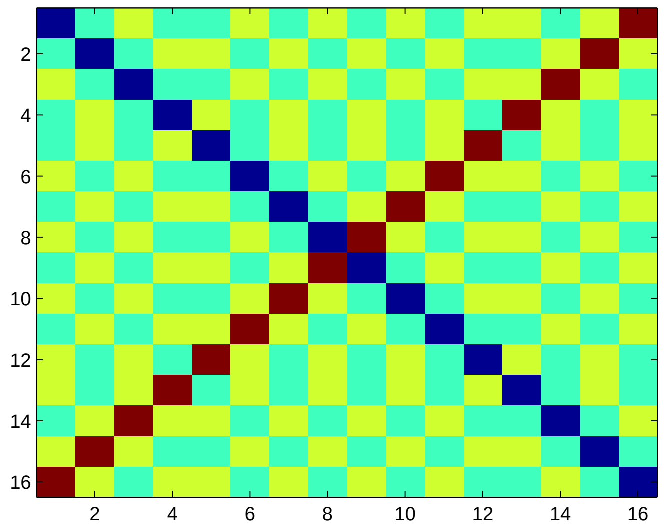

Figure 4: Picture plot of hamming distance matrix

DISCUSSION

In figure 3 we see that for a low value of SNR (low signal, high noise) that the hamming error correction does not do much better than no error detection. This is because the Hamming code can only correct single bit errors. Since the probability of multiple errors occurring is high for high noise, Hamming cannot do much better than no error correction.

In figure 3 we can also see that as the signal becomes better and the noise less that the Hamming(7,4) code outperforms no error correction. As the signal increases the magnitude of the slope for BER with hamming increases faster than no error correction.

The hamming code used in this practical was the Hamming(7,4) code. For four data bits three parity bits have to be added. This results in a 75% extra increase in the amount of bits needed. However for 256 data bits only 9 check bits have to be added, and hence this is only a 3.52% increase. Thus it can be seen that the efficiency increases as the amount of data bits per block increases. The use of error correcting codes for a larger amount of data bits is more efficient than using a error detection code and retransmitting the message.

This practical successfully investigated the impact of using a error correction code on the bit error rate on a AWGN channel.

CONCLUSION

In this practical we were successfully able to model a AWGN-like communication channel. A set of four bits in all possible permutations were sent over the channel and the bit error rate for both a hamming encoding and no encoding was plotted.

Furthermore it was found that the minimum Hamming distance between all possible code words was three. A block code with a minimum Hamming distance of three is able to detect less than 3 errors and is able to detect and correct a single incorrect bit.

From the results we can see that the use of a Hamming(7,4) code improves the bit error rate, except at a low SNR. If additional overhead is not a problem then using a Hamming encoding on a lossy channel would improve performance.

APPENDIX - SOURCE

Listed in this appendix is the Matlab source code used to generate the results

Functions:

```matlab
function [ berr ] = BERR(msgs2, snr_db )
% This function returns the bit error rate of white gaussian noise with
% SNR snr_db added to msgs2 compared to the original msgs2
    code_errs = awgn(msgs2,snr_db);

    code_errs(code_errs<0)=0;
    code_errs(code_errs>1)=1;
    code_errs = round(code_errs);

    recon_msgs = code_errs;

    [~,ratio] = biterr(recon_msgs,msgs2);

    berr = ratio + 0.000001;
end
```

```matlab
function [ berr ] = BERR74(msgs2, n, k, snr_db )
% This function returns the bit error rate of white gaussian noise with
    SNR snr_db
% added to H74 code words

    codes2 = encode(msgs2,n,k,'hamming/binary');

    code_errs = awgn(codes2,snr_db);

    code_errs(code_errs<0)=0;
    code_errs(code_errs>1)=1;
    code_errs = round(code_errs);

    recon_msgs = decode(code_errs,n,k,'hamming/binary');

    [~,ratio] = biterr(recon_msgs,msgs2);

    berr = ratio + 0.000001;
end
```

```matlab
function [ d ] = hamming_distance( byte_arr1 , byte_arr2 )
    d=sum(xor(byte_arr1 , byte_arr2));
end
```

Script to plot BER:

```
m = 3;
n = 2^m -1;
k = 4;
columns = 100; % Number of data points
trials = 50000; % Number of times to simulate
SNRdb = linspace(1,15,columns); % Range of SNR

berrs_regular = zeros(trials, columns); % Initialize berrs matrix
berrs_h74 = zeros(trials, columns);

msgs10 = 0:1:(2^k - 1);
msgs2 = de2bi(msgs10, 'left-msb');
for j = 1:trials
    % obtain bit error rate
    %We obtain both the bit error with and without using the hamming code
    berrs_regular(j,:) = arrayfun(@(e) BERR(msgs2, e), SNRdb);
    berrs_h74(j,:) = arrayfun(@(e) BERR74(msgs2, n, k, e), SNRdb);
end
berr_mean_regular = mean(berrs_regular); % take average of bit error
    rates
berr_mean_h74 = mean(berrs_h74);
% Set the graph options and plot both datasets
figure;
semilogy(SNRdb, berr_mean_regular, '-*r'); % plot on logarithmic-y plot
hold on;
semilogy(SNRdb, berr_mean_h74, '-ob');
title('Bit error rate over SNR (dB)');
xlabel('SNR (dB)');
ylabel('Bit error rate');
legend('BERR no correction','BERR with hamming(7,4)')

% Minimum Hamming distance of code words
codes = encode(msgs2,n,k,'hamming/binary');
min_distance = n;
dist_mat = zeros(2^k,2^k); % store differences in dist_mat
for i = 1:(2^k)
    for j = 1:(2^k)
        distance = hamming_distance(codes(i,:),codes(j,:));
        dist_mat(i,j) = distance;
        if i==j
            continue; % Skip if testing same row
        end;

        if distance < min_distance
            min_distance = distance;
        end
    end
end
disp(['The minimum hamming distance between code words is: ', num2str(
    min_distance)])
```

BIBLIOGRAPHY

[1] D. J. MacKay, *Information theory, inference, and learning algorithms*, vol. 7. Citeseer, 2003.

[2] T. K. Moon, "Error correction coding," *Mathematical Methods and Algorithms. Jhon Wiley and Son*, 2005.

[3] A. B. Forouzan, *Data Communications & Networking (sie)*. Tata McGraw-Hill Education, 2006.

[4] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.