Communication Systems
Experiment 2
ENG473

Shane Reynolds

September 27, 2016

Date Performed: September 26, 2016
Instructor: Dr Sina Vafi

1 Objective

Investigate the performance of cyclic codes. Specifically, the experiment focuses on the encoding and subsequent decoding of a simulated data transmission with ideal Gaussian white noise introduced over the channel. The experiment considers standard (7,4) cyclic coding, in addition to product coding using two (7,4) cyclic codes. The Bit Error Rate is recorded for various values of the signal to noise ratio as a measure of performance. The performance of both coding methods will be analysed.

2 Part A: Process & Results

A cyclic code is a subset of linear block codes, which means that it includes all zero codes and the modulo-2 sum of any two code vectors results in another code vector which is itself in the cyclic code set. A cyclic code has the further condition that a cyclic shift of the code also results in a code vector which is in the cyclic code set. Typically a cyclic code is represented as an (n, k) code where k is the number of bits of information being transmitted, along with $n-k$ check bits.

In this experiment we are implementing a (7,4) cyclic code. The length of the input message was set to 100 bits, indicating that there will be twenty five 4-bit messages passed over the transmission channel. The message was encoded using a (7,4) cyclic code, and modulated ready for transmission. The transmission simulation saw Gaussian white noise added to the channel at various Signal to Noise ratios (S/N). The received signal was decoded and the decoded message was compared to the original message. Code performance for various values of

the S/N ratio was measured using the Bit Error Rate (BER). Table 1 shows the BER for increasing values of SNR.

Table 1: Bit error rate (BER) for standard Cyclic (7,4) code

SNR (dB)	**Number of simulations**	**BER**
0.00	100	0.15190
1.00	1000	0.10250
2.00	1000	0.07050
3.00	1000	0.04460
4.00	2000	0.02480
5.00	2000	0.01180
6.00	2000	0.00430
6.50	5000	0.00240
7.00	10000	0.00130
7.50	20000	0.00065
8.00	30000	0.00032
8.50	40000	0.00013
9.00	50000	0.00005
9.25	100000	0.00003
9.50	100000	0.00002

As the S/N ratio increases, the signal swamps the noise added to the channel in the simulated transmission. We observe rapid attenuation in the BER as a result. The effects of the increase S/N can be seen in Figure 1 below. The MATLAB code used to implement the simulation can be seen in Appendix A.

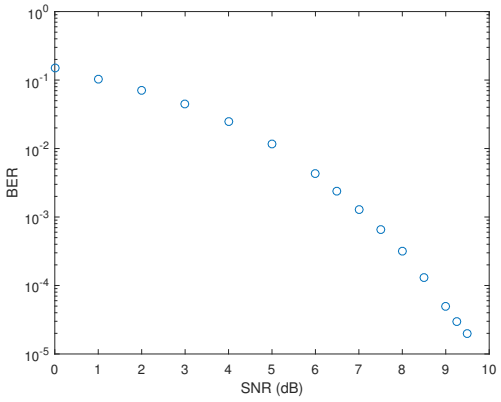

Figure 1: Performance of (7,4) Cyclic Code.

3 Part B: Process, Results & Discussion

A product code is generated using two linear block codes arranged in a matrix form. If one linear block code has n_1, k_1, and d_{min1} and the other linear block code has n_2, k_2, and d_{min2}, then according to Proakis (2002), the resulting code is an $(n_1 n_2, k_1 k_2)$ linear block code. The theoretical advantage of a product code is the increased Hamming distance. If we were using simply one of the above cyclic codes, then we would have either a Hamming distance of d_{min1} or d_{min2}, however, with the $(n_1, k_1) \times (n_2, k_2)$ product code, we get a Hamming distance of $d_{min1} \times d_{min2}$. In the context of the simulation, this increased minimum distance, should manifest itself as smaller BER. The simulation was run similarly to the previous simulation with the single cyclic coding. The resulting data from the simulation can be seen in Table 2.

Table 2: Bit error rate (BER) for Product Cyclic (7,4) × (7,4) code

SNR (dB)	**Number of simulations**	**BER**
0.0	100	0.124900
1.0	1000	0.072900
2.0	1000	0.036800
3.0	10000	0.015900
4.0	10000	0.005000
4.5	10000	0.002300
5.0	10000	0.001200
5.5	10000	0.000491
6.0	10000	0.000178
6.5	20000	0.000040
7.0	50000	0.000018
7.5	50000	0.000003
8.0	100000	
8.5	100000	

Figure 2 shows both the plotted results from the (7,4) cyclic code simulation, and the cyclic product code (7,4)×(7,4) simulation. It observed that the performance of the (7,4)×(7,4) product code is better than the simple (7,4) cyclic code. As outlined above, this is because the product code has a greater d_{min} when compared to the single cyclic code. The MATLAB code implementing the simulation can be found in Appendix B.

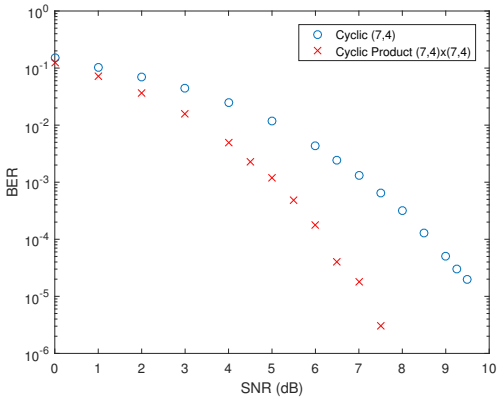

Figure 2: Performance of Cyclic (7,4) $\times$ (7,4) Product Code.

4 Conclusion

Experimental evidence was collected on encoding and transmission over a noisy channel using both a simple (7,4) cyclic code and a (7,4)$\times$(7, 4) product code. The noise was simulated using a Gaussian process. It was observed that the product code outperformed the cyclic code in terms of bit error ratio for all simulated signal to noise ratios providing evidence of superior performance should this be implemented in the field of error detection and correction.

5 References

Proakis, J. (2002). *Communication Systems Engineering.* Pretice Hall.

6 Appendix A

The MATLAB code used to implement the (7,4) cyclic code simulation.

```
% Clear the worspace and variables
clear; clc;

% Set important parameters
% Signal to noise ratio (in dB)
snr_vec = [0 1 2 3 4 5 6 6.5 7 7.5 8 8.5 9 9.25 9.5];
n = 7;
k = 4;
str_len = 100; % Length of the message.
no_blk = str_len/k; % Number of blocks with length k.
no_try_vec = [100 1000 1000 1000 2000 2000 2000 5000 10000 ...
                20000 30000 40000 50000 100000 100000];

% Defining the generator matrix
g = [1 1 0 1]; % Generator definition g(x) = x^3+x^2+1

for itr = 1:15

    snr = snr_vec(itr);
    no_try = no_try_vec(itr);
    fprintf('SNR: %d  Tries: %d',snr,no_try)

for ii = 1:no_try
    % Generate binary bitstreams with length k.
    bit_str = randi([0 1],no_blk,k);

    % A 7-bit cyclic code is generated for each of the 25 4-bit words
    for i = 1:no_blk
        msg(1,:) = bit_str(i,:);
        c(i,:) = encode(msg,n,k,'cyclic',g);
    end

    % The 25 7-bit cyclic codewords are placed into a single string
    % and BPSK modulation is applied to them
    for i = 1:no_blk
        for j = 1:n
            mux_code((i-1)*n + j) = 2*c(i,j) - 1;
        end
    end

    % The modulated signal is ready for transmission. AWGN is put
    % on the bitstream to simulate the effects of transmission
    ucode = real(awgn(mux_code,snr,'measured'));

    % Demux and a two level Quantization is undertaken on the
    % noisy 'recieved' signal
    for i = 1:no_blk
        for j = 1:n
            demux_code(i,j) = ucode((i-1)*n + j) > 0;
        end
    end
```

```
    % Decoding of 25 binary 7-bit received words after Quantization
    for i = 1:no_blk
        decoded(i,:) = decode(demux_code(i,:),n,k,'cyclic',g);
    end

    % Reordering of the decoded bitstreams
    for i = 1:no_blk
        for j = 1:k
            decoded1(i,k-(j-1)) = decoded(i,j);
        end
    end

    % We record the number of errors for each for each step in the
    % number of tries loop
    [n1,ber1] = biterr(decoded,bit_str);
    no_error(ii) = n1;
    BER(ii) = ber1;
end

% Calculate the average BER for the given parameters no_try and str_len
sum = 0;
for ii = 1:no_try
    sum = sum + no_error(ii);
end
overall_error = sum;
BER_overall = overall_error/(str_len*no_try)

clear bit_str msg c mux_code ucode demux_code no_error BER
clear decoded decoded1

end
```

7 Appendix B

The MATLAB code used to implement the (7,4) $\times$ (7,4) product cyclic code simulation.

```
% Initialising - Clearing variables and workspace
clear; clc;

% Initialising - Setting initial variables
n1 = 7;
k1 = 4;
n2 = 7;
k2 = 4;
no_try_vec = [100 1000 1000 10000 10000 10000 10000 ...
                10000 10000 20000 50000 50000 100000 100000];
snr_vec = [0 1 2 3 4 4.5 5 5.5 6 6.5 7.0 7.5 8.0 8.5];
gx1 = [1 1 0 1];
gx2 = [1 1 0 1];
str_len = 112;
no_blk = str_len/(k1*k2);
```

```
for itr = 1:14

    no_try = no_try_vec(itr);
    snr = snr_vec(itr);
    fprintf('SNR: %d  Tries: %d',snr,no_try)

    bit_str = randi([0 1],no_try,no_blk*k1*k2);

for jj = 1:no_try
    uu(1,:) = bit_str(jj,:);
    u_u = reshape(uu,k1*k2,no_blk);
    u = u_u';

    % Encoding Process (Start)
    for i = 1:no_blk
        M(1,:) = u(i,:);
        A = reshape(M,k1,k2);
        MM = A';
        for ii = 1:k2
            msg(1,:) = MM(ii,:);
            c(1,:) = encode(msg,n1,k1,'cyclic',gx2);
            code(ii,:) = c(1,:);
        end

        in_encoder2 = code';

        for ii = 1:n1
            msg2(1,:) = in_encoder2(ii,:);
            c2(1,:) = encode(msg2,n2,k2,'cyclic',gx2);
            code2(ii,:) = c2(1,:);
        end
        % Encoding Process (End)

        % Modulation & Passing through AWGN channel (Start)
        for ii = 1:n1
            ucode(ii,:) = real(awgn(2*code2(ii,:)-1,snr,'measured'))>0;
        end
        % Modulation & Passing through AWGN channel (End)

        % Decoding (Start)
        for ii = 1:n1
            decoded(ii,:) = decode(ucode(ii,:),n2,k2,'cyclic',gx2);
        end

        % Transposing to dcode each column of the codeword
        deint_codeword = decoded';

        for ii = 1:k2
            decoded2(ii,:) = decode(deint_codeword(ii,:),n1,k1,'cyclic',gx1);
        end

        % Returning the codeword to its original shape (Start)
        A = decoded2';
        B = reshape(A,1,k1*k2);
        decoded3(i,:) = B(1,:);
    end
    % Returning the codeword to its original shape (End)
```

```
    % Decoding (End)

    % Making decoded bitstreams with size of no_try x str_len (Start)
    A = decoded3';
    B = reshape(A,1,no_blk*k1*k2);
    decoded4(1,:) = B;

    bit_str2(1,:) = bit_str(jj,:);
    [n_err,ber1] = biterr(decoded4(1,:),bit_str2(1,:));
    no_error(jj) = n_err;
    BER(jj) = ber1;
end
% Making decoded bitstreams with size of no_try x str_len (End)

% Calculate Bit Error Rate (BER)
sum = 0;
for jj = 1:no_try
    sum = sum + no_error(jj);
end
overall_error = sum;
BER_overall = overall_error/(str_len*no_try)

clear uu ucode u_u u no_error msg msg2 MM M in_encoder2
clear deint_codeword decoded4 decoded3 decoded2 decoded
clear code2 code c2 c bit_str2 bit_str A B

end
```