ELEC5507 Assignment

Vanush Vaswani - 308196465
Zhaoyan Liu - 440092733
Stephen Tridgell - 309205867

May 30, 2014

Project Introduction

The goal of Error Control Coding is to encode messages for transmission with redundancy such that the receiver can correct the errors in the transmission and recover the original data. In this project, encoders and decoders will be simulated and analysed by using MATLAB. The first section involves the design and simulation of the BCH code over BSC and AWGN channels. The second section uses LDPC codes to meet the requirements of a power limited system with very high reliability. Similar simulations are run and compared with the results of the BCH code.

Section I - BCH code

Question 1

What is the generator polynomial for this code? Use MATLABs bchgenpoly function to verify your answer

The generator polynomial for a n = 31, k = 16, and t = 3 BCH code is $g(x) = x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x^1 + 1$. This was verified with

```
[gen, t] = bchgenpoly(31,16)
```

Question 2

What is the minimum distance of this code?

For a BCH code, $d_{min} \geq 2 * t + 1 = 7$, this was verified using

```
gfweight(double(gen.x), 31)
```

which gave a minimum distance calculation of 7.

Question 3

Construct the reduced syndrome lookup table for this code. You need to write a program to do this since it is difficult to do it by hand. You do not need to include the whole array in your report due to its large size. Instead, just show the sub-array consisting of the first 5 rows in your report, and include a separate text file (.txt) enumerating all the data in your electronic submission.*

Generate the full syndrome lookup table by:

```
[h g k] = cyclgen(31, double(gen.x))
trt = syndtable(h)
```

This table can be reduced in size for more efficient implementation by using the cyclic properties of the BCH code. It is calculated in the script *computeReduced.m*. The first 5 rows are:

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

The corresponding syndromes for these error patterns are:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0
0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	=	179
0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	=	539
0	0	0	0	0	1	1	0	0	0	0	1	1	0	1	=	781
0	0	0	1	0	0	0	0	0	0	1	0	1	1	1	=	2071

The rest of the array is in *reducedSyndromeTable.csv*. This contains the error pattern in binary then the decimal representation of the syndrome with the left bit as the most significant bit.

Question 4

Based on the standard array you obtained in task 3, find out the weight distribution of the coset leaders.

For the full syndrome table, or standard array coset leaders, the weight distribution is calculated as follows:

```
weights = sum(trt')
A0 = sum(weights == 0)
A1 = sum(weights == 1)
A2 = sum(weights == 2)
A3 = sum(weights == 3)
A4 = sum(weights == 4)
A5 = sum(weights == 5)
```

Gives the weight distribution of $A_0 = 1$, $A_1 = 31$, $A_2 = 465$, $A_3 = 4495$, $A_4 = 13020$ and $A_5 = 14756$ where A_i is the number of weight i errors. This is the number of possible combinations up to A_3 where $A_i = \binom{31}{i}$. Similarly the weight distribution of the reduced syndrome table is calculated. This gave the weight distribution of $A_0 = 1$, $A_1 = 1$, $A_2 = 15$, $A_3 = 145$, $A_4 = 840$ and $A_5 = 2835$. This can be verified for A_1, A_2 and A_3 as they contain 31 cycles of all words. Hence the total for A_3 is $\frac{\binom{31}{3}}{31} = 145$.

Question 5

Design and implement an encoder using the generator polynomial for this BCH code.

The encoder with the generator polynomial was implemented using the cyclic properties of the code. The message, $c(X)$, is shifted by $n - k$ and divided by the generator polynomial to get the parity check bits by $b(X) = remainder[\frac{X^{n-k}c(X)}{g(X)}]$. The parity check bits are combined with the message for a systematic code by taking $v(x) = [b(x)c(x)]$. The function *polBCHencoder.m* implements the generator polynomial method of decoding the BCH code.

Question 6a

Use MATLAB defined functions (eg. the decode function) to decode the BCH code.

This decoder is implemented using MATLABs function *decode()*. The matrix *trt* is the full syndrome decoding table calculated above using the *syndtable()* function. This decode method is based on the linear block properties of the code. The syndromes

are calculated and the most likely codeword is decoded to by looking up the minimum weight. The implementation is in the function *matlabBCHdecode.m*.

Question 6b

Design and implement a decoder using the syndrome decoding table in Task 3 for the BCH code.

The syndrome can be obtained by multiplying received codeword with the transposed parity-check matrix H. The corresponded error pattern is found by performing a look up in the trt table calculated above. The corrected code is then the codeword plus the error pattern. Our implementation is shown in the function *syndLookupDecode.m*. The memory overhead of the decoder can be reduced by significantly reducing the size of the lookup table by using cyclic properties of the code. This was implemented using the reduced lookup table calculated previously. The recieved vector is shifted until its syndrome is recognized in the table. The error locations are then looked up in the table and used to correct the codeword. The implementation of this decoder is in *syndReducedLookupDecode.m*

Question 6c

Design and implement a decoder using the Berlekamps iterative procedure.

The implementation of Berlekamp's iterative procedure below used different notation to the textbook specified in *http://www.mathworks.com.au/help/comm/ug/error-detection-and-correction.html* allowing easier implementation of the procedure. See the function *berlekamp_decode.m* for the matlab code.

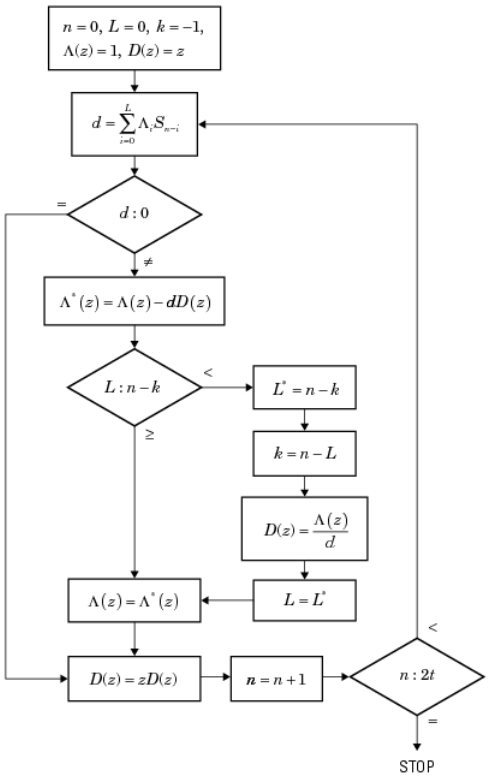

Figure 1: Flow chart of berlekamps algorithm from www.mathworks.com.au/help/comm/ug/error-detection-and-correction.html

This differs from the textbook method in its labeling of variables and a slight modification on the calculation of σ_ρ. It instead computes the this as the variable D with d_ρ already divided through. This allows for easier implementation in software when multiplying the polynomials.

Sound analysis

Question 7

Simulate the implemented BCH encoder and decoder (using ANY method in Part 6) using the attached wave file (austinpowers.wav) in a BSC channel for different transition probabilities. Discuss the impact of changing different transition probability values.

The channel was simulated using the function *playAudioOverBSC.m*. Different transition probabilities were input into the function. With a transition probability of below 0.05 the sound could be heard with a little noise audible. As p increased above 0.15 the noise increased to the level where it could no longer be clearly heard. Above 0.2 it was difficult to hear anything at all other than noise.

The plots for different p-values are shown below as a qualitative indicator of noise levels.

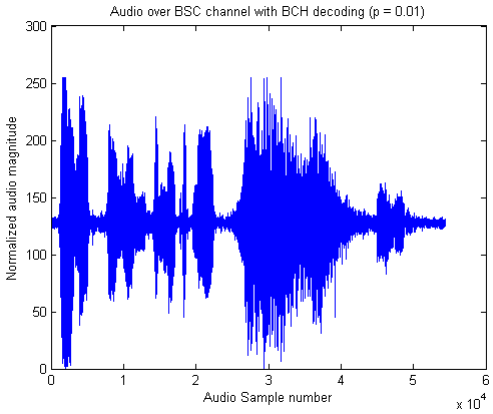

Figure 2: BCH Audio over BSC (p = 0.01)

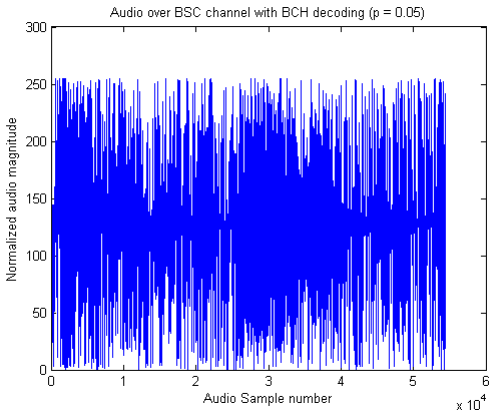

Figure 3: BCH Audio over BSC (p = 0.05)

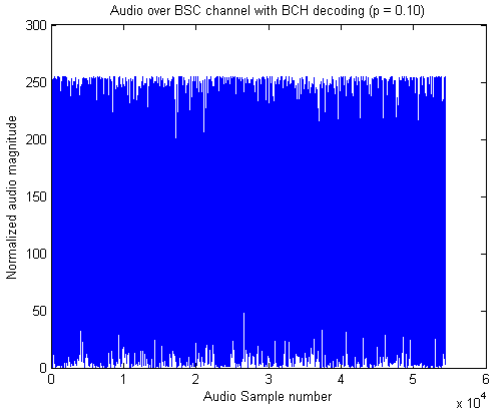

Figure 4: BCH Audio over BSC (p = 0.1)

BER analysis

Question 8a and 8b

Simulation over BSC Simulate the BCH code in a BSC channel (you are allowed to use MATLAB defined functions) and plot the BER versus transition probability for coded and uncoded systems on the same graph. Plot BER versus $\frac{E_b}{N_0}$ *for coded and uncoded systems on the same graph by assuming that the SNR (*$= \frac{E_b}{N_0}$ *) is related to the transition probability for the coded system via* $X_{dB,coded} = 10\log_{10}(\frac{[Q^{-1}(p)]^2}{R})$ *and that for uncoded system is* $X_{dB,uncoded} = 20\log_{10}(Q^{-1}(p))$*, where* $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{+\infty} e^{-\frac{z^2}{2}} dz$ *and* $0 < p < 0.5$.
Simulation over AWGN channel with BPSK modulation $\{-1, 1\}$ *Simulate the BCH code in an AWGN channel (you are allowed to use MATLAB defined functions) and plot the BER versus the signal to noise ratio (SNR) for a BPSK coded and uncoded systems on the same graph by using a hard-decision demodulator and binary decoder.*

The implementation of the simulation is in *bch_ber_simulation.m*. This produced the following plots. The full range to 10^{-6} BER was not calculated as there were memory limitations. Instead specific points of interest we calculated such as 10^{-4}, 10^{-5} and 10^{-6} in addition to these plots for the subsequent table.

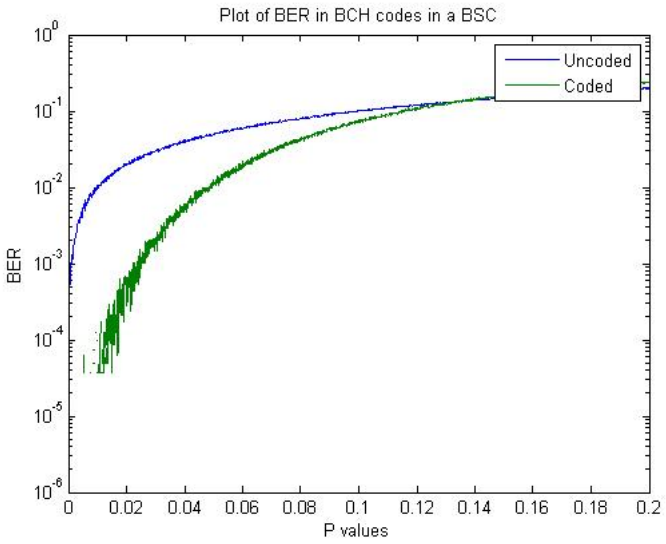

Figure 5: Bit Error Rate vs. transition probability for BCH over BSC

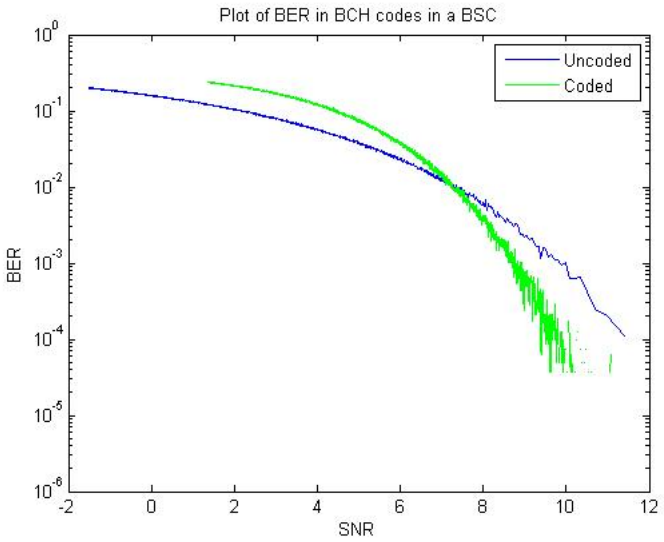

Figure 6: Bit Error Rate vs. SNR for BCH over BSC

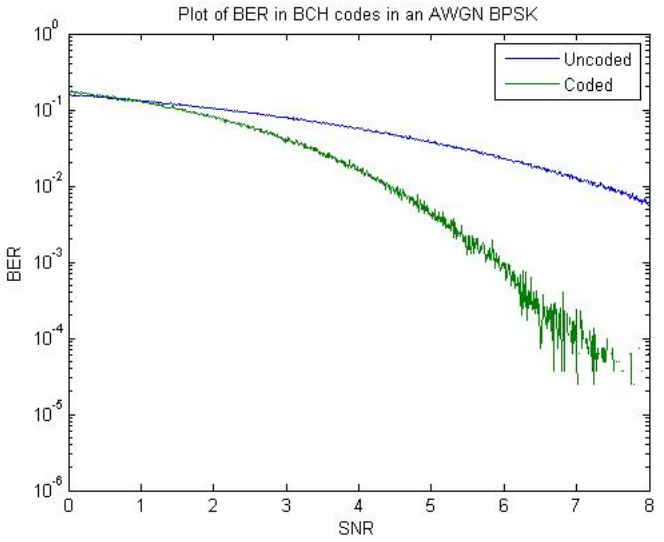

Figure 7: Bit Error Rate vs. SNR for BCH over AWGN

Question 8c

Draw a table detailing the coding gain for BER$= [10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$ *by reading the differences between the BER curves for BPSK coded and uncoded systems which have been obtained from simulations in part (a) and (b).*

The following values were obtained from the plot where possible. For the smaller

values of 10^{-6} and 10^{-5} the points in the table were obtain by running it for individual points. For a transition probability of 0.0037 in the BSC a BER of approximately 10^{-6} was obtained for the coded signal. A transition probability of 10^{-6} obviously gives a BER of $10^{-}6$ for the uncoded signal. This gives $X_{db,coded} = 11.42$ dB and $X_{db,uncoded} = 13.54$ dB giving a coding gain of 2.11 dB. Similarly a transition probability of 0.007 in the BSC a BER of approximately 10^{-6} was obtained for the coded signal. A transition probability of 10^{-5} obviously gives a BER of $10^{-}5$ for the uncoded signal. This gives $X_{db,coded} = 10.68$ dB and $X_{db,uncoded} = 12.596$ dB giving a coding gain of 1.92 dB.

For a SNR of 8.3 dB in the AWGN a BER of approximately 10^{-6} was obtained for the coded signal. A SNR of 13.5 dB gave a BER of 10^{-6} for the uncoded signal. This gives a coding gain of 5.2 dB. Similarly, For a SNR of 7.5 dB in the AWGN a BER of approximately 10^{-5} was obtained for the coded signal. A SNR of 12.5 dB gave a BER of 10^{-5} for the uncoded signal. This gives a coding gain of 5 dB. Additionally a SNR of 11.1 dB gave approximately a BER of 10^{-4} for the uncoded signal and a SNR of 9.8 dB gave a BER of 10^{-3}. The rest of the values were obtained from the plot.

BER	Coding Gain (Hard decision)	Coding Gain (BSC)
10^{-2}	3	0
10^{-3}	3.8	1.3
10^{-4}	4.1	1.7
10^{-5}	5	1.92
10^{-6}	5.2	2.11

Table 1: Coding gain for BCH

Question 8d

Find the asymptotic coding gain when $\frac{Eb}{N0}$ is very large from the formula which is given in the lecture notes and compare with simulation results.

Using the formula $G = 10\log(R(t+1))$ in lecture 5 for large $\frac{Eb}{N0}$ where $R = 16/31$ and $t = 3$ results in a coding gain of $G = 3.1482$ dB. Comparing this to the simulation results the BSC and the hard decision decoded AWGN channel strangely do not match. Additionally the asymptotic coding gain is less than the hard decision decoded value at 10^{-6} BER. We are unsure why this is the case as our simulation results have been checked.

Section II - LDPC

You are an engineer whose job is to design and analyze the performance of error control codes for different clients. Client 2 requires a code for satellite transmission of digital TV. The satellite is power limited and very high reliability is required (as close to Shannon capacity as possible). Low decoding complexity is desired, but is not essential.

Question 1

Design the code (e.g. type of code, code parameters, and other relevant information) and justify its design

To meet these requirements an LDPC code is used

For client 2, it is clear that an LDPC code will be necessary and these can provide performance that is close to the Shannon limit for an AWGN channel, compared to other modern codes. However, it is known that this is the case for very long block codes, which require computationally complex. An existing solution for digital satellite television is present in the DVB-S2 standard, which uses LDPC as part of concatenated code with BCH coding [1]. However, the drawback of this scheme is the high decoding complexity, though this can be reduced somewhat by the special structure of the code . Another factor to consider is the power limitation of the satellite, since a large LDPC block code will also entail high coding complexity, more hardware gates and thus more power. An alternative code choice is the (1152, 2304) LDPC block code used in IEEE-802.16-2005, also known as WiMax. This code is also irregular, which is known to exhibit superior performance over regular LDPC codes.

Another aspect of LDPC is that it requires iterative decoding. In general the BER improves with the number of iterations. To model the low complexity desired, the number of iterations used throughout the report is 5.

- Code Type: LDPC
- Code Size: (1152, 2304)
- Code Rate: 0.5
- Density: 0.002748

Sound analysis

Question 2

Simulate the chosen code using the attached wave file (austinpowers.wav) in a BSC channel for different transition probabilities (you are allowed to use MATLAB defined functions. Discuss the difference in sound quality compared to the BCH code in Section I, for different transition probabilities.

The code was simulated for the transition probabilities p = $\{0.01, 0.05, 0.1\}$. As a qualitative tool, plots of the waveform are included. Figures 1-3 provide a visual indicator of the noise.

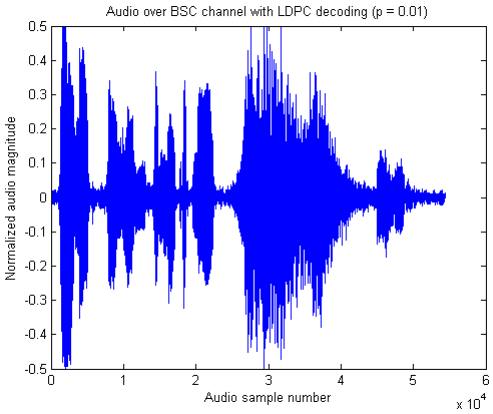

Figure 8: LDPC Audio over BSC (p = 0.01)

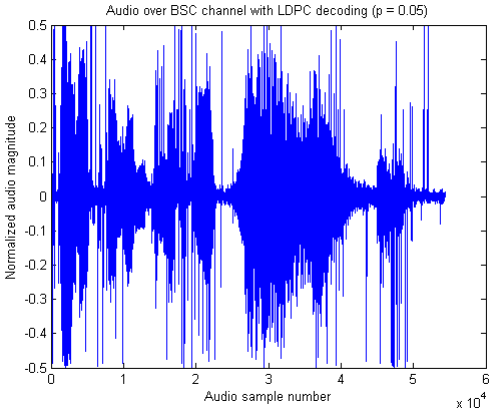

Figure 9: LDPC Audio over BSC (p = 0.05)

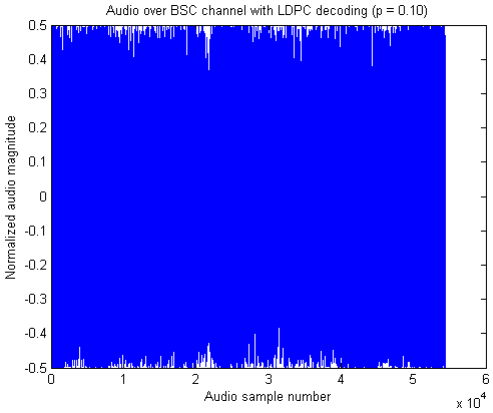

Figure 10: LDPC Audio over BSC (p = 0.1)

As the bit errors increase due to the transition probability, the noise becomes more prevalent. However, it is improved relative to BCH in the case of p = 0.05, showing LDPC has superior performance in this regard compared with BCH. However, there is not much difference in the case of p = 0.1, showing that the code has reached it's performance limitation for the particular channel, which is due to effective hard-decision decoding that is involved in a BSC channel.

Question 3a

Simulate the chosen code in a BSC channel (you are allowed to use MATLAB defined functions) and plot the BER versus transition probability for coded and uncoded systems on the same graph. Plot BER versus $\frac{E_b}{N_0}$ *for coded and uncoded systems on the same graph by assuming that the SNR (*$= \frac{E_b}{N_0}$*) is related to the transition probability for the coded system via* $X_{dB,coded} = 10\log_{10}(\frac{[Q^{-1}(p)]^2}{R})$ *and that for uncoded system is* $X_{dB,uncoded} = 20\log_{10}(Q^{-1}(p))$*, where* $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{+\infty} e^{-\frac{z^2}{2}} dz$ *and* $0 < p < 0.5$.

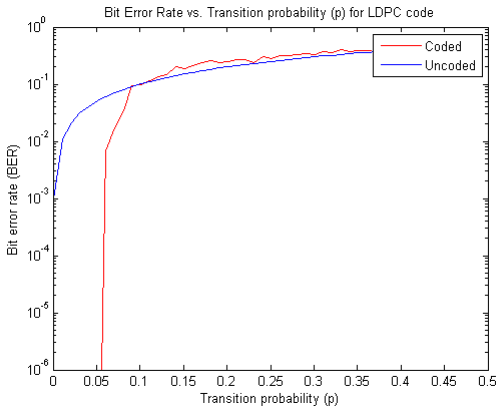

Figure 11: BER vs transition probability for LDPC over BSC

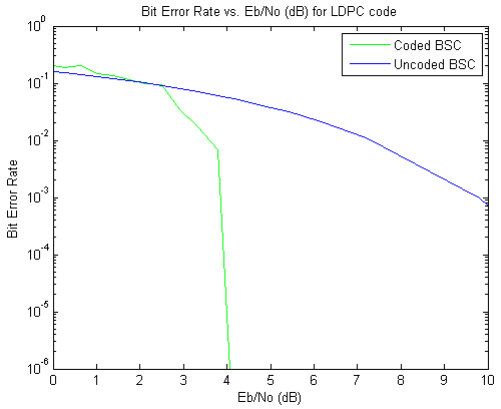

Figure 12: BER vs EbNo for LDPC over BSC

BER analysis

Question 3b

Simulate the chosen code in an AWGN channel (you are allowed to use MATLAB defined functions) and plot the BER versus the signal to noise ratio (SNR) for a BPSK coded and uncoded systems on the same graph by using a hard-decision demodulator and binary decoder. Simulate the chosen code in an AWGN channel and plot the BER versus SNR on the same graph for a BPSK coded system by using a soft-decision decoder.

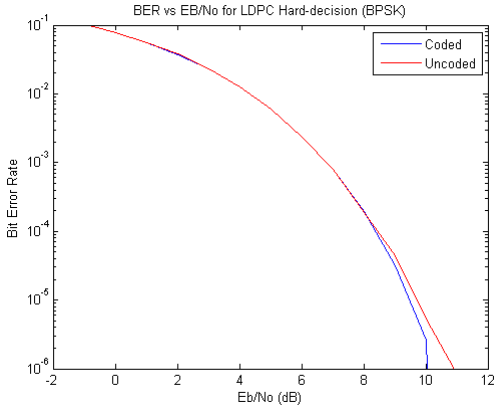

Figure 13: BER vs EbNo for Hard-decision demod and decoding (LDPC - AWGN)

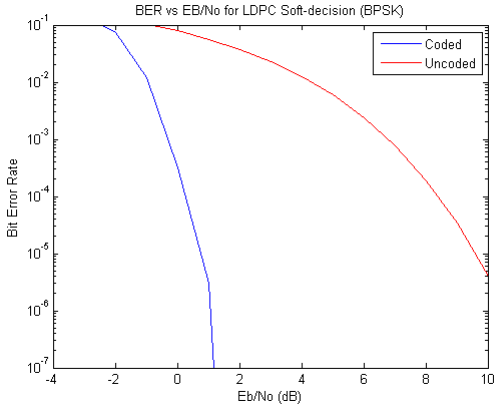

Figure 14: BER vs EbNo for Soft-decision demod and decoding (LDPC - AWGN)

Question 3c

Draw a table detailing the coding gain for BER$= [10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$ *by reading the differences between the BER curves for BPSK coded and uncoded systems which have been obtained from simulations in part (a) and (b).*

BER	Coding Gain (Hard decision)	Coding Gain (Soft decision)	Coding Gain (BSC)
10^{-2}	0	6	3.5
10^{-3}	0	7.8	5.5
10^{-4}	0	8	NA
10^{-5}	0	9.25	NA
10^{-6}	0	9	NA

Table 2: Coding gain for LDPC

Due to lack of simulation time, the coding gain for 10^-4, 10^-5, and 10^-6 was unable to be calculated for BSC; however, it is very high. The highest performance gain is seen in soft-decision decoding. This is due to the algorithm implemented by the MATLAB Communications Toolbox LDPC decoder: it uses the iterative message passing algorithm which makes use of soft information in the form of log-likelihood ratios. On the other hand, the result for hard-decision decoding seems erroneous due to the algorithm requiring soft input. For the BSC channel, soft input was simulated by running the BSC-corrupted data through a noiseless BPSK mod/demod, allowing log-likelihood ratios to be outputted.

Question 4

Discuss the advantages/disadvantages of the code chosen in this section with the BCH code in Section 1 (eg. complexity, coding gain, efficiency)

BCH codes have significantly lower complexity than the LDPC as BCH has the properties that it is a cyclic, linear block code. This allows various decoding methods depending on resources available. BCH can also be made systematic allowing for simpler decoding methods. LDPC codes are additionally large in size generally, compared to the BCH codes. This means that there is greater latency in the delivery of data, depending on code rate. LDPC however, performs significantly better than BCH in terms of coding gain with it's ability to approach the Shannon capacity. A further drawback of LDPC is the "error floor", where the BER does not improve even though the SNR does. One technique to fix this is to concatenate LDPC and BCH, allowing BCH to fix errors caused by the error floor. In general, there is a tradeoff involved in using LDPC; high complexity, low efficiency but higher coding gain. Nowadays, it is worth it because of the extreme power limitation in applications such as deep space communications.

Conclusion

In this report, error-control codes were analysed and simulated in MATLAB. First, a BCH code was analysed and characteristics determined. A decoder implemented using Berlekamp's algorithm. It exhibited fair error correcting performance (as quantified by coding gain) compared to an uncoded BPSK in an AWGN channel. Next, an LDPC code was selected for a hypothetical client to satisfy low complexity and near-Shannon capacity performance for satellite communications. A small number of iterations (5) was selected and the encoder and decoder simulated in MATLAB using the message-passing algorithm. It exhibited an even higher coding gain than the BCH code assuming the soft-decision assumptions were met. Both these codes show the fundamental im-

portance of error correcting codes in modern communications, allowing low bit error rates at lower EbNo, increasing reliability with the tradeoff of high complexity, though this drawback can be offset with modern ASIC technology and the special structure of particular codes.

References

[1] Alberto Morello and Vittoria Mignone. Dvb-s2: the second generation standard for satellite broad-band services. *Proceedings of the IEEE*, 94(1):210–227, 2006.

Appendix - Matlab code

File bch_ber_simulation.m

```
% simulations section 1) part 8)
% Find the BER over a BCH code
pValues = [0.005 0.002 0.001];
repetitions = 10000000;

msg = rand(length(pValues)*repetitions, 16) > 0.5;
errors = rand(length(msg), 31) < ...
        repmat(reshape(repmat(pValues,repetitions,1),1,[])',1,31);
bch_encoded = encoder(msg);

bch_decoded = matlabBCHdecode(mod(bch_encoded + errors, 2));
bitErrs = reshape((sum(mod(bch_decoded' + msg',2))/16)', repetitions,
    []);
BER = sum(bitErrs)/length(bitErrs); % take the average
errors = rand(length(msg), 16) < ...
        repmat(reshape(repmat(pValues,repetitions,1),1,[])',1,16);
bitErrs = reshape((sum(mod(errors',2))./16)', repetitions, []);
BERUncoded = sum(bitErrs)/length(bitErrs);

figure()
semilogy(fliplr(pValues), fliplr([BERUncoded; BER]));
title('Plot␣of␣BER␣in␣BCH␣codes␣in␣a␣BSC');
legend('Uncoded', 'Coded');
ylim([10^-6 10^0]);
xlabel('P␣values');
ylabel('BER');

% Q(x) = 0.5 * ( 1 - erf(x/sqrt(2)))
% Q^(-1)(x) = sqrt(2)*erfinv(1 - 2x)

Xuncoded = 20.*log10(sqrt(2).*erfinv(1 - 2.*pValues));
Xcoded = Xuncoded - 10.*log10(16/31); % R = code Rate = k/n

figure()
semilogy(Xuncoded, BERUncoded, Xcoded, BER, '-g');
title('Plot␣of␣BER␣in␣BCH␣codes␣in␣a␣BSC');
legend('Uncoded', 'Coded');
ylim([10^-6 10^0]);
xlabel('SNR');
ylabel('BER');

clear all; % otherwise run out of memory
% section 1) q 8)b)

snrValues = 0:0.01:8;
repetitions = 5000;

% SNR = 10*log10((1/ampl)^2), ampl = 10^(-SNR/20)
msg = rand(length(snrValues)*repetitions, 16) > 0.5;
bch_encoded = encoder(msg);
noise = randn(length(bch_encoded), 31).* ...
      repmat(reshape(repmat(10.^(-snrValues./20),repetitions,1) ...
            ,1,[])',1,31);
signal = 1 - 2.*bch_encoded;
bch_decoded = matlabBCHdecode((signal + noise) < 0); % Hard decision on
     0
bitErrs = reshape((sum(mod(bch_decoded' + msg',2))./16)', repetitions,
    []);
```

```
BER = sum(bitErrs)/length(bitErrs); % take the average

noise = randn(length(msg), 16).* ...
      repmat(reshape(repmat(10.^(-snrValues./20),repetitions,1) ...
             ,1,[])',1,16);
signal = 1 - 2.*msg;
decoded = (signal + noise) < 0;
bitErrs = reshape((sum(mod(decoded' + msg',2))./16)', repetitions, []);
BER2 = sum(bitErrs)/length(bitErrs); % take the average

figure()
semilogy(snrValues, [BER2; BER]);
title('Plot␣of␣BER␣in␣BCH␣codes␣in␣an␣AWGN␣BPSK');
legend('Uncoded', 'Coded');
ylim([10^-6 10^0]);
xlabel('SNR');
ylabel('BER');
```

File computeReduced.m

```
load trt
% go through all errors
allCycles = trt;
uniqueTable = [];
for i = 1:31
    tmp = [trt(:,i:end) trt(:,1:(i-1))];
    allCycles(:,:,i) = tmp;
    uniqueTable = unique([uniqueTable; allCycles(:,:,i)], 'rows');
end
redTable = [];
for index = 1:31
    [~, intA, intB] = intersect(allCycles(:,:,index), uniqueTable, '
    rows');
    [~, I] = sort(intA);
    redTable = [redTable intB(I)];
end
uniqueIndexs = ones(length(uniqueTable),1);
for r = redTable'
    if sum(uniqueIndexs(r)) == 31
        uniqueIndexs(r(2:end)) = zeros(30,1);
    elseif sum(uniqueIndexs(r)) ~= 1
        error('huh?')
    end
end
uniqueIndexs(1) = 1; % all zero gets deleted
reducedTable = uniqueTable(find(uniqueIndexs),:);
[~, iA, iB] = intersect(reducedTable, trt, 'rows');
[~, I] = sort(iA);
syndromes = iB(I) - 1;
save('reducedTable.mat','syndromes', 'reducedTable');
```

File matlabBCHdecode.m

```
function decoded_data = matlabBCHdecode(data_to_decode)
load trt
bchPol = [1 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1];
decoded_data = decode(data_to_decode,31,16,'cyclic/fmt', bchPol, trt);
```

File polBCHencoder.m

```
function codeword = polBCHencoder(msg)
genPol = [1 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1];
c = [msg zeros(1,15)];
```

```
[~, b] = gfdeconv(fliplr(c), fliplr(genPol));
b = [zeros(1,15 - length(b)) fliplr(b) zeros(1,16)];
codeword = b + [zeros(1,15) msg];
```

File syndReducedLookupDecode.m

```
function decoded_data = syndReducedLookupDecode(data_to_decode)
load reducedTable
H = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0
     0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0
     0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0
     0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1
     0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1
     0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 1
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0
     0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1
     0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0
     0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1];
for index = 1:31
    s = [data_to_decode(index:end) data_to_decode(1:(index-1))]*H';
    s = bin2dec(int2str(mod(s,2)));
    if sum(syndromes == s) > 0
        errorInd = find(syndromes == s);
        errorLoc = reducedTable(errorInd,:);
        y = mod([data_to_decode(index:end) ...
            data_to_decode(1:(index-1))] + errorLoc, 2);
        decoded_data = [y((end - index+2):end) y(1:end -index + 1)];
        decoded_data = decoded_data(16:end);
        return
    end
end
error('HELP! cant find!')
```

File berlekamp_decode.m

```
function [data, decoded_data, error_locations] = ...
                    berlekamp_decode(data_to_decode)

% Parameters of n = 31, k = 16, t = 3 BCH
n = 31;
t = 3;
k = 16;
m = 5; % 2^5 = 32

% GF field vars
zero = gf(0, m);
one = gf(1, m);
alpha = gf(2, m);

codeword = gf(data_to_decode, m);

% compute the syndromes
S =  codeword*(alpha.^([1:2*t]'*fliplr(0:30)))';

L = 0;
k = -1;
sigma = [one gf(zeros(1,(2*t - 1)), m)];
D = [zero one gf(zeros(1,(2*t - 2)), m)];
```

```
for n = 0:(2*t - 1)
    discrepency = fliplr(S((n-L + 1):(n + 1)))*sigma(1:(L + 1))';
    if discrepency ~= 0
        sigma_star = sigma - discrepency*D;
        if L < n - k
            L_star = n - k;
            k = n - L;
            D = sigma/discrepency;
            L = L_star;
        end
        sigma = sigma_star;
    end
    D = [zero D(1:(end-1))];
end


% Find roots of sigma
rootsOfSigma = [];
rootToTry = alpha.^(1:31);
for index = 1:length(rootToTry)
    % if sum is zero then is root
    if sigma*(rootToTry(index).^(0:(length(sigma) - 1)))' == zero
        rootsOfSigma = [rootsOfSigma index];
    end
end
degree = find(sigma ~= zero);
degree = degree(end) - 1;
error_locations = zeros(1,length(data_to_decode));
if length(rootsOfSigma) < degree
    % disp('WARNING: Cannot correct more than 3 errors')
    decoded_data = data_to_decode;
    data = decoded_data(16:end); % assuming a systematic code
    return
end
error_locations(rootsOfSigma) = ones(1,length(rootsOfSigma));
decoded_data = mod(data_to_decode + error_locations,2);
data = decoded_data(16:end); % assuming a systematic code
```

File encoder.m

```
function codeword = encoder(msg)
G = [ 1 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
      1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
      0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
      0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
      0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
      0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
      1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
      1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
      1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
      1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
      0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
      0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
      0 0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1];
codeword = mod(msg*G, 2);
```

File playAudioOverBSC.m

```
function playAudioOverBSC(pVal)
```

```
[data, Fs] = wavread('austinpowers.wav', 'native');
data = de2bi(data);
% add a row of zeros as it is not divisible by 16
data = double(reshape([data; zeros(1,8)],[],16));
errors = rand(length(data),31) < pVal;

encoded_data_no_error = encoder(data);
encoded_data = mod(encoder(data) + errors,2);

decoded_data = reshape(matlabBCHdecode(encoded_data),[], 8);
% remove the last row of zeros again
decoded_data = bi2de(decoded_data(1:end-1,:));
soundsc(double(decoded_data),Fs);
figure
plot(double(decoded_data));
plot_title = sprintf('Audio over BSC channel with BCH decoding (p = %.2
    f)', pVal);
xlabel('Audio Sample number')
ylabel('Normalized audio magnitude')
title(plot_title)
```

File syndLookupDecode.m

```
function msg = syndLookupDecode(codeword)
load trt
H = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0
     0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0
     0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0
     0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1
     0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1
     0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 1
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0
     0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1
     0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0
     0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1];
syndrome = bi2de(mod(codeword * H',2),'left-msb');
errLocations = trt(1+syndrome,:);
correctedcode = mod(errLocations+codeword,2);
msg = correctedcode(:,16:end);
```

File binary_to_wav.m

```
function binarywav = binary_to_wav(data)
    d = char(data+48);
    d1 = bin2dec(d);
    binarywav = d1./255;
    binarywav = binarywav - 0.5;
end
```

File plot_ebno.m

```
% Plots coded and uncoded Eb/No given transition probabilities
% and the code rate. Formulas given in project notes
function plot_ebno(transition_prob, bers, uncoded_bers, code_rate)

    % Replace zero's with episolon for plot-friendliness
    bers(bers==0) = 10e-10;
    uncoded_bers(uncoded_bers==0) = 10e-10;
```

```
    % Determine Eb/No for coded/uncoded
    ebno_coded =  10*log10 (  (qfuncinv(transition_prob).^2) / (
    code_rate));
    ebno = 20*log10(qfuncinv(transition_prob));

    % Plot Eb/No vs for coded/uncoded
    figure
    semilogy(ebno_coded(2:end-1), bers(2:end-1), 'g');
    hold on
    semilogy(ebno(2:end-1), uncoded_bers(2:end-1));
    legend('Coded BSC', 'Uncoded BSC');
    xlabel('Eb/No (dB)');
    ylabel('Bit Error Rate')
    xlim([0 10])
    ylim([10^-6 10^0]);
    title('Bit Error Rate vs. Eb/No (dB) for LDPC code');
    hold off

end
```

File simulate_ldpc_awgn_hard.m

```
% Simulate LDPC over AWGN with BPSK
% with hard decision demod and hard decision decoder

close all
clear all

% Import LDPC parity matrix
load ldpc_h
ldpc_h = sparse(ldpc_h);

% Generate EbNo's
ebnos = -2:12;

% LDPC Encoder/Decoder
hEnc = comm.LDPCEncoder(ldpc_h);

% Hard decision by default
hDec = comm.LDPCDecoder(ldpc_h, 'MaximumIterationCount', 5);

hMod = comm.PSKModulator(2);
hDemodHard = comm.PSKDemodulator(2, 'BitOutput',true,...
            'DecisionMethod','Hard decision');

hError = comm.ErrorRate;
hError_uncoded = comm.ErrorRate;

bers = [];
bers_uncoded = [];

% For each eb/no, encode using LDPC, modulate, demodulate,
% decode, then determine error rate. Do the same for uncoded
numErrors = 0;
i = 0;

for e = ebnos

    % Create noisy channel using EbNo.
    hChan = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (SNR
    )', 'SNR', e);
```

```
    while i <  1000

        % Generate data
        data = logical(randi([0 1], 1152, 1));

        % Encode data
        encodedData = step(hEnc, data);

        % Modulate encoded data
        modData = step(hMod, encodedData);
        % Modulate uncoded data
        modData_uncoded = step(hMod, data);

        % Add AWGN noise according to EbNo.
        noisySignal = step(hChan, modData);
        noisySignal_uncoded = step(hChan, modData_uncoded);

        % Demod
        receivedSignal = step(hDemodHard, noisySignal);
        receivedSignal_uncoded = step(hDemodHard, noisySignal_uncoded);

        %  The output is bits. Need to convert to log-likelihood ratio
        % for LDPC
        % More positive = likely to be 0
        % More negative = likely to be 1
        receivedSignal(receivedSignal==0) = 10000;
        receivedSignal(receivedSignal==1) = -10000;

        %receivedSignal_uncoded(receivedSignal_uncoded>0) = 0;
        %receivedSignal_uncoded(receivedSignal_uncoded<0) = 1;

        % LDPC decode
        dataDec = step(hDec, receivedSignal);

        % Determine error rate
        errorstats = step(hError, data, dataDec);
        errorstats_uncoded = step(hError_uncoded, data, logical(
    receivedSignal_uncoded));

        i = i + 1

    end

    bers  = [bers errorstats(1) ];
    bers_uncoded = [bers_uncoded errorstats_uncoded(1)];

    i = 0;
    % Reset error stats
    reset(hError);
    reset(hError_uncoded);

end

bers(bers == 0) = 10e-15;
figure
semilogy(ebnos, bers);
hold on
semilogy(ebnos, bers_uncoded, 'r');
legend('Coded', 'Uncoded');
ylim([10^-6 10^-1])
xlabel('Eb/No (dB)')
ylabel('Bit Error Rate')
```

```
title('BER vs Eb/No for LDPC Hard-decision (BPSK)')
%fprintf('Error rate       = %1.2f\nNumber of errors = %d\n', ...
%      errorstats(1), errorstats(2))
```

File simulate_ldpc_bsc.m

```
% Simulate LDPC over BSC
% Part II, Question 3a)
% Author: Vanush Vaswani

close all
clear all

% Import LDPC parity matrix
load ldpc_h
ldpc_h = sparse(ldpc_h);

% Generate transition probabilities
ps = [0.0000000001 0.0000001 0.000001 0.00001 0.001:0.01:0.5];

% Generate data
data = logical(randi([0 1], 1152, 1));

% LDPC Encoder/Decoder
hEnc = comm.LDPCEncoder(ldpc_h);
hDec = comm.LDPCDecoder(ldpc_h, 'MaximumIterationCount', 5);

% Go through perfect BPSK mod/demod (no noise) - this is because the
    LDPC Decoder
% requires log-likelihood ratios
hMod = comm.PSKModulator(2, 'BitInput',true);
hDemod = comm.PSKDemodulator(2, 'BitOutput',true, ...
            'DecisionMethod','Approximate log-likelihood ratio');
hError = comm.ErrorRate;

% Generate arrays of BER
bers = [];
i = 0;

disp ('Running simulation ...')
for p = ps

    hError = comm.ErrorRate;

    % Encode data
    data_enc = step(hEnc, data);

    % BSC channel
    data_enc_bsc = bsc(double(data_enc), p);
    data_bsc = bsc(double(data), p);

    % Mod/demod
    data_enc_bsc_mod = step(hMod, data_enc_bsc);
    data_enc_bsc_demod = step(hDemod, data_enc_bsc_mod);

    % Decode
    data_dec = step(hDec, data_enc_bsc_demod);

    % Get BER
    errorstats = step(hError, data, data_dec);

    bers = [bers errorstats(1)];
end
```

```
disp('Plotting␣BER␣vs␣p␣...')
plot_ber(ps, bers);
disp ('Plotting␣BER␣vs␣EBNo␣...')
plot_ebno(ps, bers, ps, 1);
```

File wav_to_binary.m

```
function wavbinary = wav_to_binary(filename)
    [x, Fs] = wavread(filename, 'native');
    x1 = x - min(x);
    s = dec2bin(x);
    wavbinary = double(s) - 48;
end
```

File plot_ber.m

```
% Given an array of tranisition probbilities and
% error rates for a code, plot them
function plot_ber(transition_prob, bers)
    % Plot transition probability vs BER

    bers(bers==0) = 10e-10;
    figure
    semilogy(transition_prob, bers,'r');
    hold on
    xlabel('Transition␣probability␣(p)');
    ylabel('Bit␣error␣rate␣(BER)');
    semilogy(transition_prob, transition_prob);
    legend('Coded', 'Uncoded');
    ylim([10^-6 10^0]);
    title('Bit␣Error␣Rate␣vs.␣Transition␣probability␣(p)␣for␣LDPC␣code'
    )
    hold off


end
```

File qpsk_test.m

```
load ldpc_h
ldpc_h = sparse(ldpc_h);
ebno = 0
   for e = ebno
        hEnc = comm.LDPCEncoder(ldpc_h);
        hMod = comm.PSKModulator(2, 'BitInput',true);
        hChan = comm.AWGNChannel(...
                'NoiseMethod', 'Signal␣to␣noise␣ratio␣(Eb/No)','EbNo',e
    );
        hDemod = comm.PSKDemodulator(2, 'BitOutput',true,...
                'DecisionMethod','Approximate␣log-likelihood␣ratio',
    ...
                'Variance', 1/10^(hChan.SNR/10));
        hDec = comm.LDPCDecoder(ldpc_h);
        hError = comm.ErrorRate;
        for counter = 1:10
            counter
          data           = logical(randi([0 1], 1152, 1));
          encodedData    = step(hEnc, data);
          modSignal      = step(hMod, encodedData);
          receivedSignal = step(hChan, modSignal);
          demodSignal    = step(hDemod, receivedSignal);
          receivedBits   = step(hDec, demodSignal);
```

```
          data == receivedBits
          errorStats     = step(hError, data, receivedBits);
        end
        fprintf('Error␣rate␣␣␣␣␣␣␣=␣%1.8f\nNumber␣of␣errors␣=␣%d\n',
    ...
          errorStats(1), errorStats(2))
   end
```

File simulate_ldpc_awgn_soft.m

```
% Simulate LDPC over AWGN with BPSK
% Hard decision demodulator and soft-decision decoder

close all
clear all

% Import LDPC parity matrix
load ldpc_h
ldpc_h = sparse(ldpc_h);

% Generate EbNo's
ebnos = -3:10;

% LDPC Encoder/Decoder
hEnc = comm.LDPCEncoder(ldpc_h);

% Hard decision by default
hDec = comm.LDPCDecoder(ldpc_h, 'MaximumIterationCount', 5, '
    DecisionMethod', 'Soft␣decision');

hMod = comm.PSKModulator(2);
hDemodHard = comm.PSKDemodulator(2, 'BitOutput',true,...
            'DecisionMethod', 'Approximate␣log-likelihood␣ratio');

hError = comm.ErrorRate;
hError_uncoded = comm.ErrorRate;

bers = [];
bers_uncoded = [];

% For each eb/no, encode using LDPC, modulate, demodulate,
% decode, then determine error rate. Do the same for uncoded
numErrors = 0;
i = 0;

for e = ebnos

    % Create noisy channel using EbNo.
    hChan = comm.AWGNChannel('NoiseMethod', 'Signal␣to␣noise␣ratio␣(SNR
    )', 'SNR', e);

    while i <  10000

        % Generate data
        data = logical(randi([0 1], 1152, 1));

        % Encode data
        encodedData = step(hEnc, data);

        % Modulate encoded data
        modData = step(hMod, encodedData);
        % Modulate uncoded data
        modData_uncoded = step(hMod, data);
```

```
        % Add AWGN noise according to EbNo.
        noisySignal = step(hChan, modData);
        noisySignal_uncoded = step(hChan, modData_uncoded);

        % Demod
        receivedSignal = step(hDemodHard, noisySignal);
        receivedSignal_uncoded = step(hDemodHard, noisySignal_uncoded);

        receivedSignal_uncoded(receivedSignal_uncoded>0) = 0;
        receivedSignal_uncoded(receivedSignal_uncoded<0) = 1;

        % LDPC decode
        dataDec = step(hDec, receivedSignal);

        dataDecThreshold = dataDec;
        dataDecThreshold(dataDecThreshold>0) = 0;
        dataDecThreshold(dataDecThreshold<0) = 1;

        % Determine error rate
        errorstats = step(hError, data, logical(dataDecThreshold));
        errorstats_uncoded = step(hError_uncoded, data, logical(
    receivedSignal_uncoded));

        i = i + 1

    end

    bers  = [bers errorstats(1) ];
    bers_uncoded = [bers_uncoded errorstats_uncoded(1)];

    i = 0;
    % Reset error stats
    reset(hError);
    reset(hError_uncoded);

end

bers(bers < 10e-10) = 10e-15;
bers_uncoded(bers_uncoded < 10e-10) = 10e-15;

figure
semilogy(ebnos, bers);
hold on
semilogy(ebnos, bers_uncoded, 'r');
legend('Coded', 'Uncoded');
ylim([10^-7 10^-1])
xlabel('Eb/No␣(dB)')
ylabel('Bit␣Error␣Rate')
title('BER␣vs␣EB/No␣for␣LDPC␣Soft-decision␣(BPSK)')
%fprintf('Error rate       = %1.2f\nNumber of errors = %d\n', ...
%      errorstats(1), errorstats(2))
```

File simulate_ldpc_wav.m

```
% Simulate LDPC encoding of audio
% Author: Vanush Vaswani

% Import LDPC parity matrix
load ldpc_h
ldpc_h = sparse(ldpc_h);

bsc_p = 0.1;
```

```
% Get wav data
data = wav_to_binary('austinpowers.wav');
data_t = data';

% Convert data to column vector
data_col = data_t(:);

k = 1152;

num_cols = floor(numel(data_col)/k);

% Clip data to an integer number of messages (K = 1152)
data_col = data_col(1:k*num_cols);

% Messages to columns
data_col = logical(vec2mat(data_col, num_cols));
[nrow, ncol] = size(data_col);

% LDPC Encoder/Decoder
hEnc = comm.LDPCEncoder(ldpc_h);
hDec = comm.LDPCDecoder(ldpc_h,'MaximumIterationCount', 20);

% Go through perfect BPSK mod/demod - this is because the LDPC Decoder
% requires log-likelihood ratios
hMod = comm.PSKModulator(2, 'BitInput',true); % Don't we want (2 ...
    for a BPSK?
hDemod = comm.PSKDemodulator(2, 'BitOutput',true, ...
            'DecisionMethod','Approximate log-likelihood ratio');
hError = comm.ErrorRate;


received_matrix = zeros(k, num_cols);

disp ('Running simulation ...')
% For each message (frame of raw audio), generate code word. Then mod/
    demod, then decode
for i = 1:ncol
    disp(i);
    % Encode
    data_encoded = step(hEnc, data_col(:,i));

    % Go through BSC (add bit errors)
    data_encoded_bsc = bsc(double(data_encoded), bsc_p);

    errors = data_encoded == data_encoded_bsc;
    error_rate = length(find(errors == 0))/numel(errors)

    %data_encoded_bsc = data_encoded;

    % Modulate
    data_encoded_bsc_mod = step(hMod, logical(data_encoded_bsc));
    % Demodulate
    data_encoded_bsc_demod = step(hDemod, data_encoded_bsc_mod);

    % Decode
    received_bits = step(hDec, data_encoded_bsc_demod);

    % Store in rx matrix
    received_matrix(:,i) = received_bits;
    errorStats  = step(hError, data_col(:,i), received_bits);
end
```

```
fprintf('Error␣rate␣␣␣␣␣␣␣=␣%1.2f\nNumber␣of␣errors␣=␣%d\n', ...
      errorStats(1), errorStats(2))
rxdata = received_matrix';
rxdata = rxdata(:);
rxdata = vec2mat(rxdata, 8);
rxwav = binary_to_wav(rxdata);

fprintf('Plotting␣corrected␣audio␣with␣BSC␣p␣=␣%d\n', bsc_p)
plot(rxwav)
plot_title = sprintf('Audio␣over␣BSC␣channel␣with␣LDPC␣decoding␣(p␣=␣
    %.2f)', bsc_p);
xlabel('Audio␣sample␣number')
ylabel('Normalized␣audio␣magnitude')
title(plot_title)
sound(rxwav, 11025);
```