Abdul Basit Anees
21600659
EEE 432

Telecommunications II

Project 1

Abdul Basit Anees

21600659

EEE 432

Abdul Basit Anees
21600659
EEE 432

Introduction

In this project, we will study potential benefits of channel coding to combat random errors introduced during transmission over a channel. Then, we discuss decoding and performance analysis of convolutional codes. We are supposed to pick a convolutional code with rate k/n = 1/3 where k = 1 and n = 3. The constraint length L was chosen as 4. Further details of the convolutional code chosen are discussed in the sections below.

Part I

Encoder Block Diagram

Following are the generators for the parity bits.

```
g1 = [1 1 0 1];
g2 = [1 0 0 1];
g3 = [1 1 1 1];
```

The block diagram for the encoder is shown below.

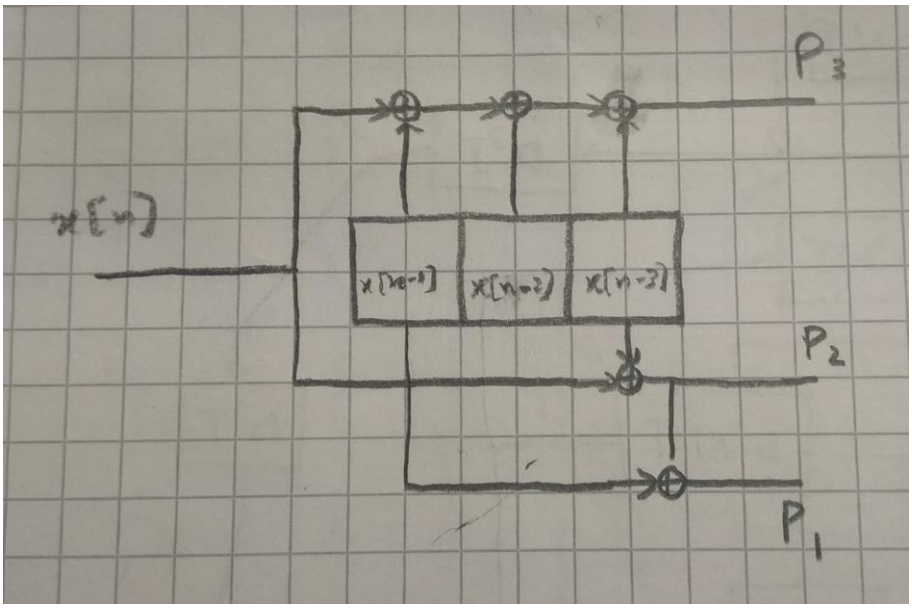

Figure 1:Encoder Block Diagram

State Flow Diagram

A code was written, which computes the state transitions and outputs depending on the constraint length L and generator vectors. Using that, the state flow diagram is obtained as follows, which will be used to obtain the transfer function.

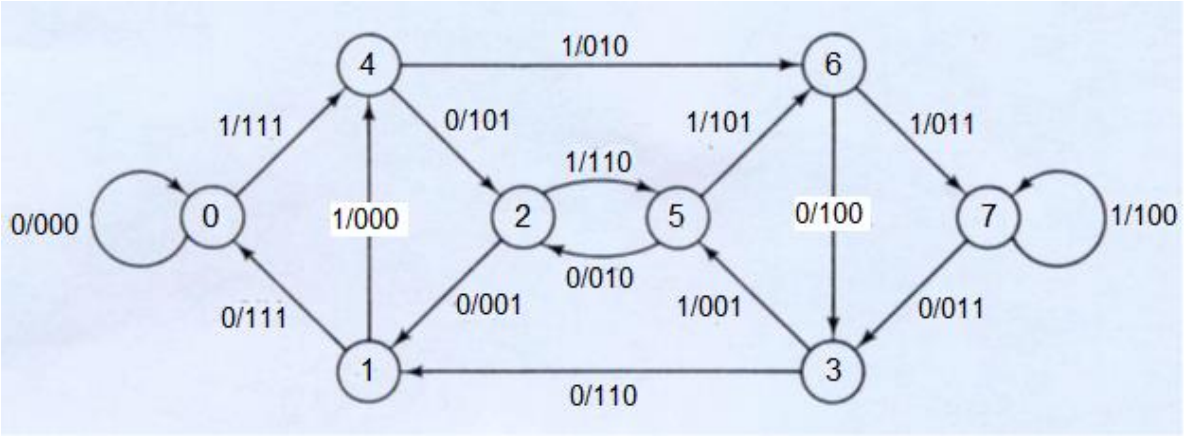

Figure 2: State Flow Diagram

Transfer Function

To compute the transfer function, the following state diagram is used to obtain state equations.

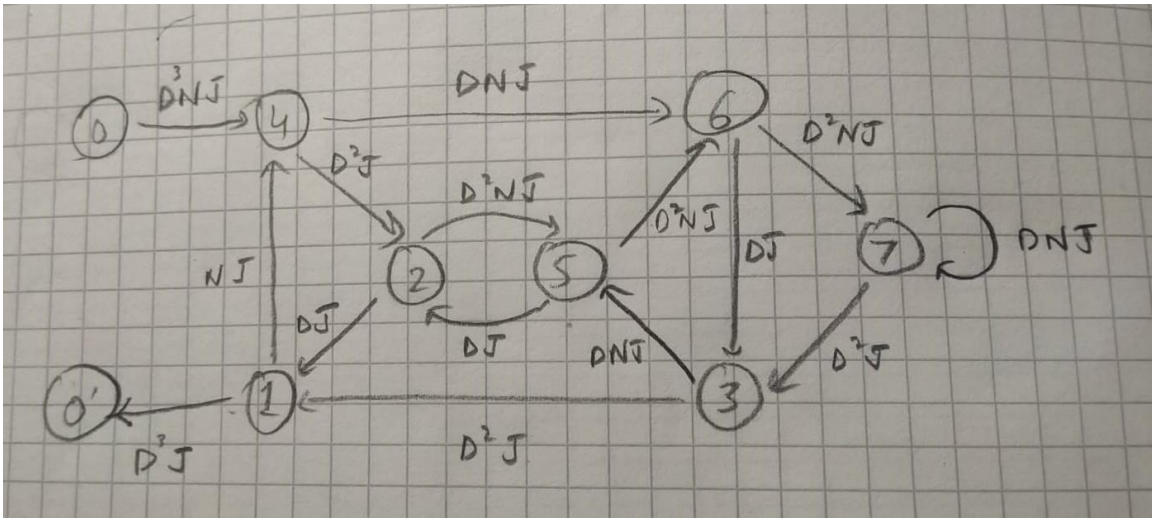

Figure 3: State Flow Diagram for transfer function

Following are the obtained state equations, where D,N,J are dummy variables and the power of D represents the weight of output of the ith Trellis section and the power of N represent the number of ones in the input (0 or 1 for k = 1).

Abdul Basit Anees
21600659
EEE 432

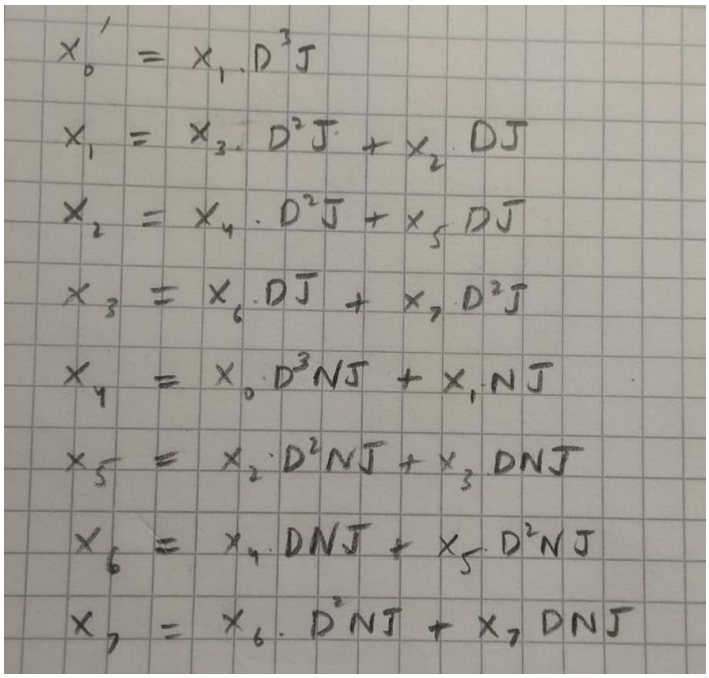

Figure 4: State Flow Equations

These state equations are solved using Matlab to obtain the following transfer function.

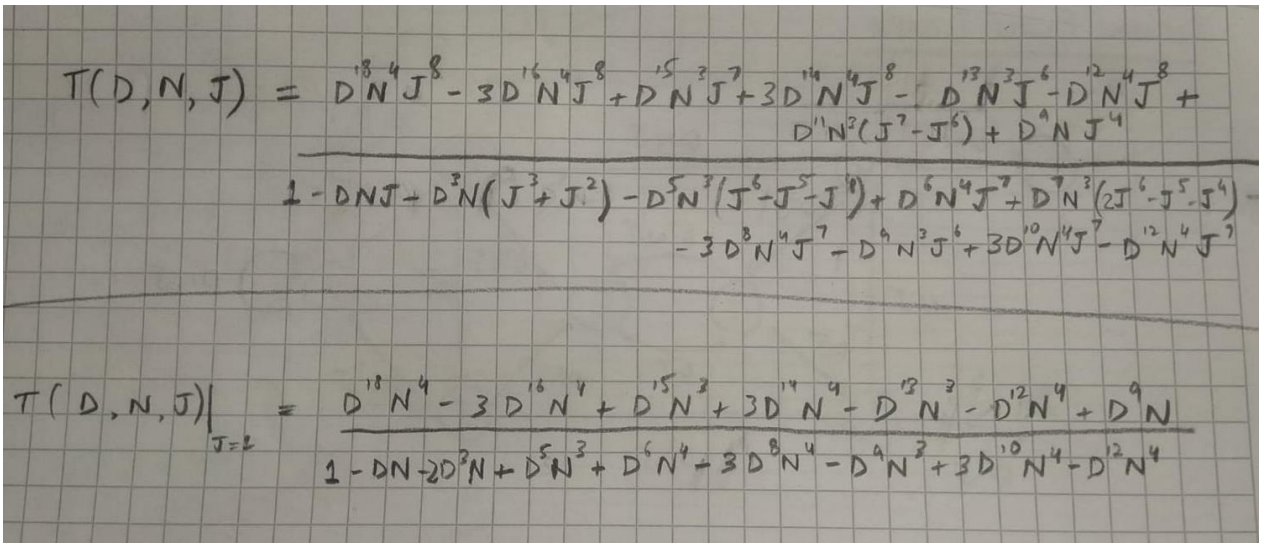

Figure 5: Transfer Function

Abdul Basit Anees
21600659
EEE 432

Part II

In this part, we implement the Viterbi decoding algorithm in Matlab and simulate the error performance over different types of channels. Codewords of length 1000 were used and the bit error was averaged of 50 different codewords for each simulation.

Binary Symmetric Channel

We can see that, using the Binary Symmetric channel (BSC), with increase in the value of p, the bit error increases linearly (It is zero for low values of p) on the logarithmic scale until it reaches the error rate of about 50% and converges.

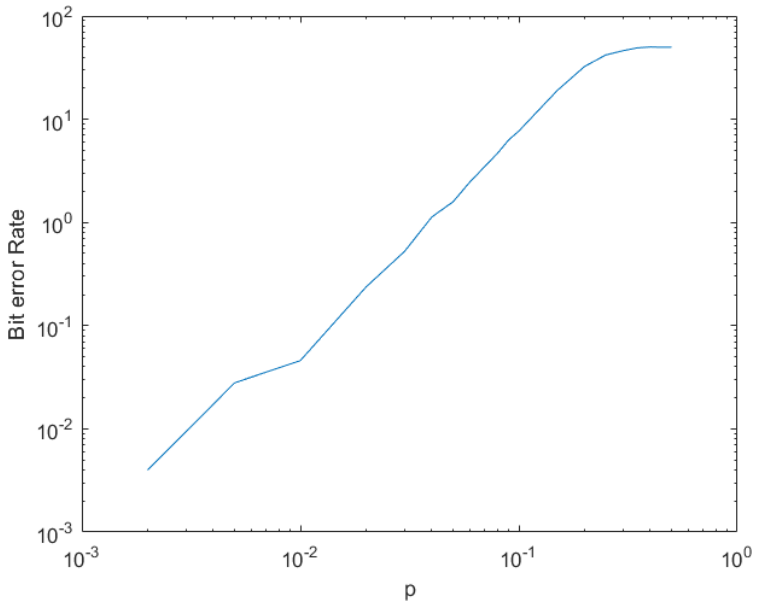

Figure 6: Error rate for BSC

AWGN Channel with HDD and SDD

For the AWGN channel, we can see that the bit error rate decreases as SNR is increased. For low SNRs, the error rate is about 50% whereas for high SNRs the error rate is 0%. We also see that SDD performs better than HDD. This is because information is lost in HDD while quantizing information in one bit.

Abdul Basit Anees
21600659
EEE 432

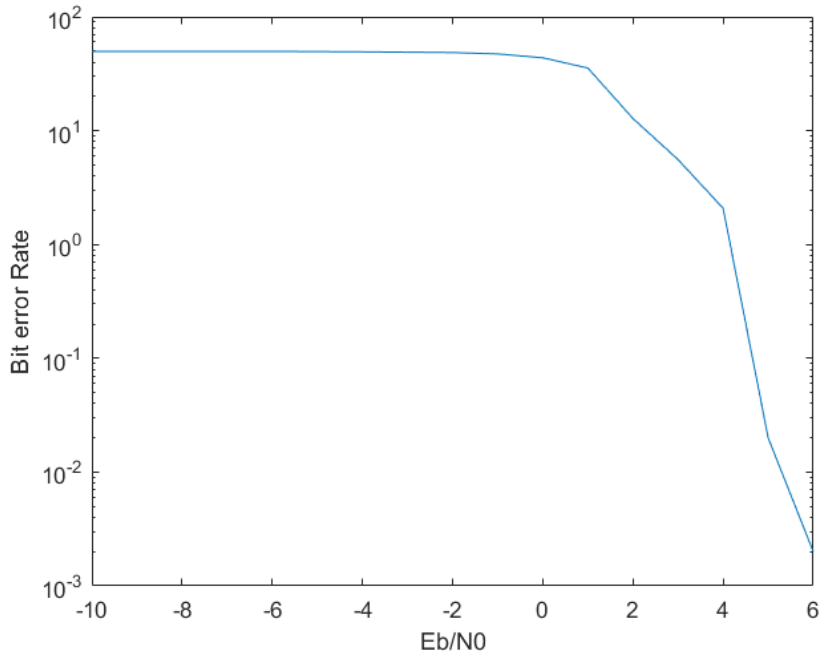

Figure 7: AWGN channel with HDD

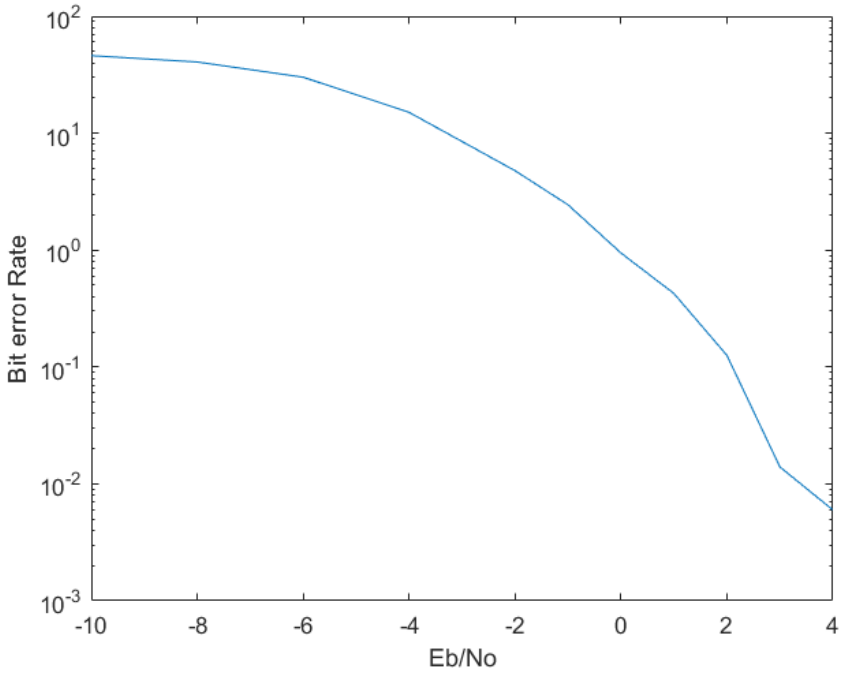

Figure 8: AWGN channel with SDD

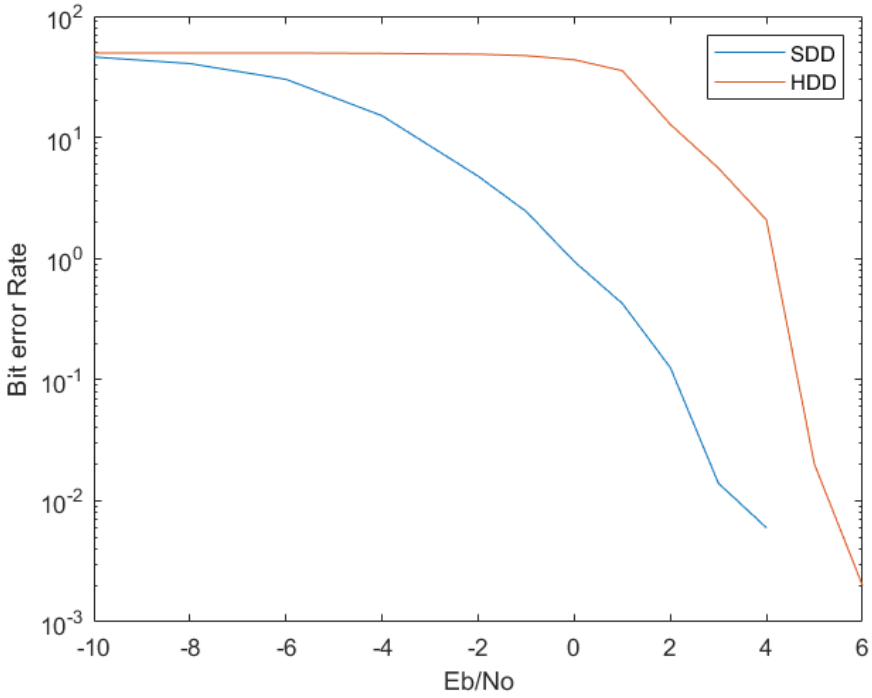

Figure 9: Comparison of AWGN channel SDD and HDD

AWGN Channel SDD Quantized

We can see that by quantization some information is lost as is discussed before. This causes a decrease in the performance of convolutional codes. SDD with quantization with higher bits (4 in this case) performs better than quantization with lower bits (2).

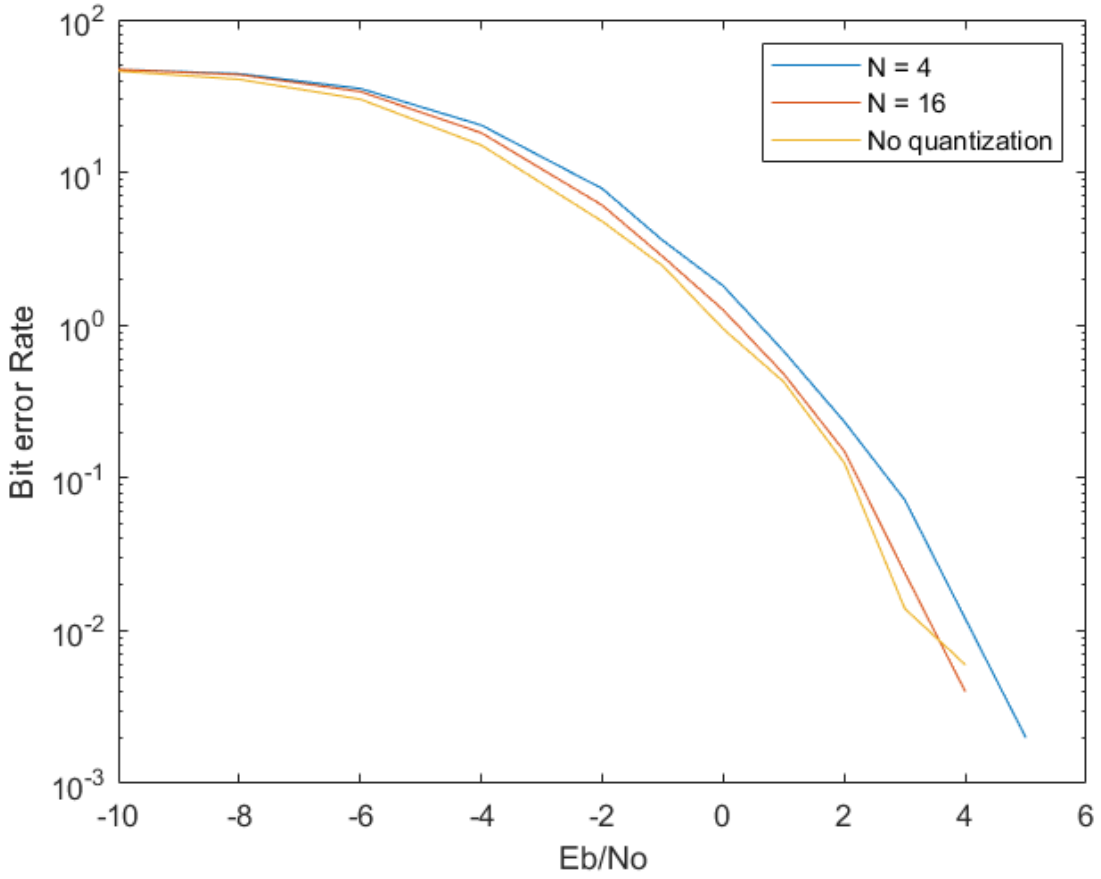

Figure 10: Effects of quantization on SDD of convolutional codes

Catastrophic code

```
g1 = [1 0 1 0];
g2 = [1 1 0 0];
g3 = [1 0 0 1];
```

It has output of 000 in a self-loop at state 7 (or 111). Therefore, it has a bad error performance for some codewords which causes very high error for those codewords and therefore resulting in higher mean bit error. It can be seen that the catastrophic codes have high bit error rates than the normal codes.

Abdul Basit Anees
21600659
EEE 432

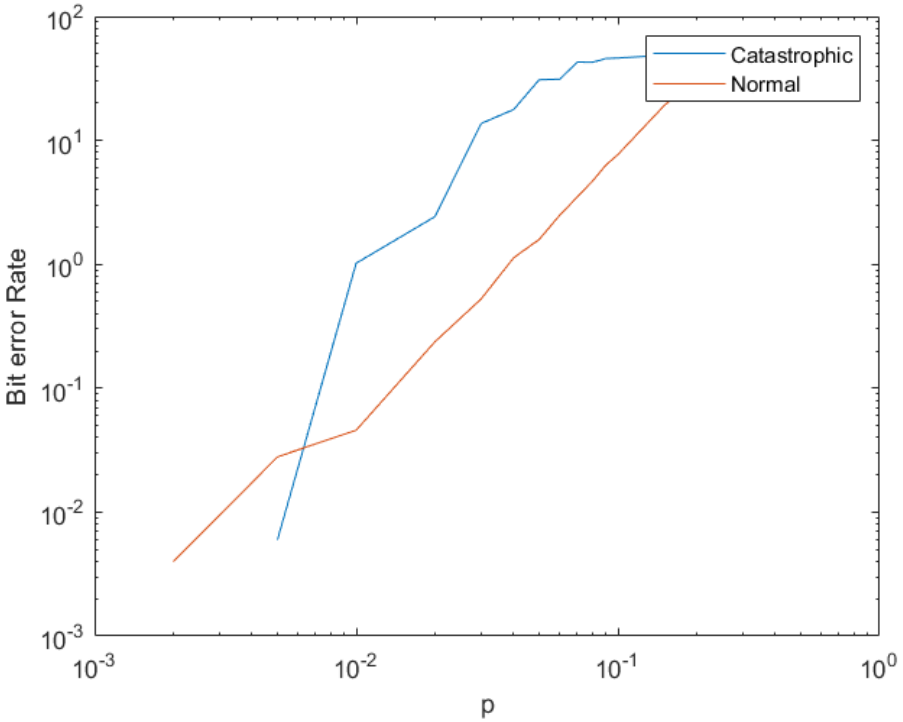

Figure 11: Performance of catastrophic code in BSC

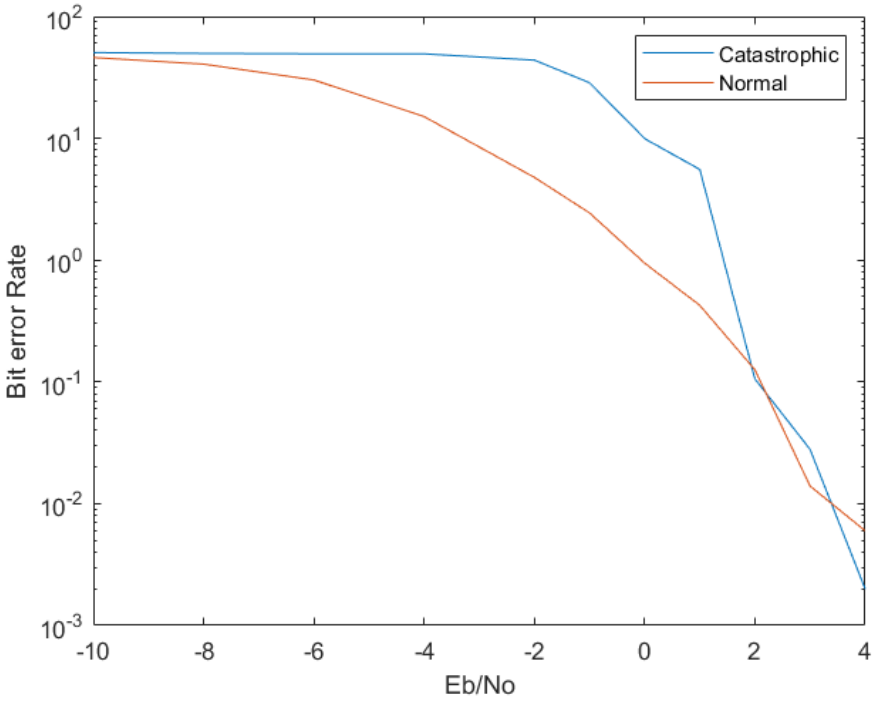

Figure 12: Performance of catastrophic code in AWGN

Abdul Basit Anees
21600659
EEE 432

Appendix

```
% The code is dynamic for any constraint length L which
depends on the length
% of generator vectors. Currently implemented for k = 1
%% Non-Catastrophic code

% parity bit vectors
g1 = [1 1 0 1]; % 13
g2 = [1 0 0 1]; % 9
g3 = [1 1 1 1]; % 15
L  = length( g1);

% Create state array and outputs for all possible inputs
trellisDict = trellisDictionary(L, g1, g2, g3);
% trellisDict.outputs(:, 2^L) % Output of Self loop

%% Implement Viterbi decoder assuming perfect receiver and
channel

% nBits = 1000;
% codeword = genRandomCodeword(nBits, L);
% tx = encodeConvolutional(codeword, g1, g2, g3);
% rx = tx;
% decodedBits = ViterbiDecoder(rx, trellisDict, L, 'HDD');
% bitErrorRate = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);

%% Simulate a binary symmetric channel
trials = 50;
nBits  = 1000;
p = [0, 0.001, 0.002, 0.005, 0.008 0.01:0.01:0.1,
0.15:0.05:0.5];
bitErrorRate = zeros(length(p), 1);
for i = 1:length(p)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        rx = bsc(tx, p(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
```

```
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end

ErrorRates.bscHDD = bitErrorRate;
% figure()
% plot(p, bitError);
figure()
loglog(p, ErrorRates.bscHDD);
xlabel('p')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN uncoded

trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = codeword;
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        rx = rx > 0;
        decodedBits = rx;
        error(j) = 100 * sum(codeword ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNUncoded = bitErrorRate;
figure()
semilogy(SNR_db, ErrorRates.AWGNUncoded);
xlabel('Eb/N0')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN channel with HDD
trials = 50;
```

```
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        rx = rx > 0;
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNHDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNHDD);
hold on
semilogy(SNR_db, ErrorRates.AWGNUncoded);
legend('coded', 'uncoded')
xlabel('Eb/N0')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN channel with SDD
trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
```

```
        rx = AWGN(tx, N0(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'SDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNSDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNSDD);
hold on
semilogy(SNR_db, ErrorRates.AWGNUncoded);
legend('coded', 'uncoded')
xlabel('Eb/No')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN channel using
quantization with SDD

trials = 50;
nBits  = 1000;
N = [4 16];
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 2);
for k = 1:length(N)
    for i = 1:length(bitErrorRate)
        disp(i)
        error = zeros(trials,1);
        for j = 1:trials
            codeword = genRandomCodeword(nBits, L);
            tx = encodeConvolutional(codeword, g1, g2, g3);
            tx(tx==0) = -1;
            rx = AWGN(tx, N0(i));
            rx = UniformQuantizer(rx, N(k));
            decodedBits = ViterbiDecoder(rx, trellisDict,
L, 'SDD');
            error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
        end
        bitErrorRate(i, k) = mean(error);
```

```
    end
end
ErrorRates.AWGNQSDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNQSDD(:,1));
hold on
semilogy(SNR_db, ErrorRates.AWGNQSDD(:,2));
hold on
semilogy(SNR_db, ErrorRates.AWGNUncoded);
legend('N = 4', 'N = 16', 'uncoded');
xlabel('Eb/No')
ylabel('Bit error Rate')
%% Catastrophic code

% parity bit vectors
g1 = [1 0 1 0]; %10
g2 = [1 1 0 0]; %12
g3 = [1 0 0 1]; %9
L  = length( g1);

% Create state array and outputs for all possible inputs
trellisDict = trellisDictionary(L, g1, g2, g3);
% trellisDict.outputs(:,16)

%% Simulation  of catastrophic code over BSC

trials = 50;
nBits  = 1000;
p = [0, 0.001, 0.002, 0.005, 0.008 0.01:0.01:0.1,
0.15:0.05:0.5];
bitErrorRate = zeros(length(p), 1);
for i = 1:length(p)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        rx = bsc(tx, p(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
```

```
    bitErrorRate(i) = mean(error);
end
ErrorRates.bscCatHDD = bitErrorRate;

figure()
loglog(p, ErrorRates.bscCatHDD);
hold on
loglog(p, ErrorRates.bscHDD);
legend('Catastrophic', 'Normal')
xlabel('p')
ylabel('Bit error Rate')

%% Simulation  of catastrophic code over AWGN with SDD

trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'SDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNCatSDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNCatSDD);
hold on
semilogy(SNR_db, ErrorRates.AWGNSDD);
legend('Catastrophic', 'Normal')
xlabel('Eb/No')
ylabel('Bit error Rate')
```

```
function trellisDict = trellisDictionary(L, g1, g2, g3)
    % This function creates state transitions for a
specific constraint
    % length and also the outputs corresponding to each
input at each state
    % transition
    nStates = 2 ^ (L - 1);
    trellisDict.nStates = nStates;
    states = cellstr( dec2bin(0 : nStates - 1));
    states = split(states, '');
    states = str2double(states(:, 2 : L));
    states = [states; states];
    input = [zeros(nStates, 1); ones(nStates, 1)];
    block = [input states];
    newStates = block(:, 1 : end-1);
    trellisDict.new = bin2Dec(newStates);
    trellisDict.old = bin2Dec(states);
    trellisDict.inputs = input;
    stateTrans = [bin2Dec(states) bin2Dec(newStates)];
    trellisDict.stateTrans = stateTrans;

    output1 = sum(block .* g1, 2);
    output2 = sum(block .* g2, 2);
    output3 = sum(block .* g3, 2);
    output = [output1 output2 output3];
    output = mod(output, 2);
    outputSDD = output;
    outputSDD(outputSDD==0) = -1;
    trellisDict.outputs = output';
    trellisDict.outputsSDD = outputSDD';
end


function dec = bin2Dec(bin)
    n = size(bin);
    n = n(2);
    vec = 2.^[n-1:-1:0];
    dec = sum(bin.*vec, 2);
end


function codeword = genRandomCodeword(nBits, L)
    bits  = randi( 2, nBits, 1) - 1;
```

```
    codeword  = [bits' zeros(1, L-1)]; % L zeros sent to
bring state to 0 in the end
%     codeword  = [zeros(1, L-1) sentBits]; % Add L zeros
to left to simulate shift register 0 initially
end

function encoded = encodeConvolutional(bits, g1, g2, g3)
    L = length(g1);
    bits = [zeros(1, L-1) bits]; % Add L zeros to left to
simulate shift register 0 initially
    % parity bits
    p1 = conv(bits, g1, 'valid');
    p2 = conv(bits, g2, 'valid');
    p3 = conv(bits, g3, 'valid');
    % transmission signal
    encoded = [p1; p2; p3];
    encoded = mod(encoded, 2);
end

function rx = bsc(tx, p)
    % Simulates a binary symmetric channel
    sizetx = size(tx);
    nBits = max(sizetx);
    nParities = min(sizetx);
    nRx = nBits * nParities;
    nOnes = round(nRx * p);
    noise = [ones(1, nOnes) zeros(1, nRx - nOnes)];
    shuffledIndex = randperm(nRx);
    noise = noise(shuffledIndex);
    noise = reshape(noise, nParities, nRx/nParities);
    rx = mod((tx + noise), 2);
end


function rx = AWGN(tx, N0)
    % Simulates an AWGN channel
    std      = sqrt(N0 / 2);
    noise    = std .* randn(size(tx));
    rx = tx + noise;
end
```

Abdul Basit Anees
21600659
EEE 432

```
function decodedBits = ViterbiDecoder(rx, trellisDict, L,
type)
    nStates = trellisDict.nStates;
    dataLength = size(rx);
    stateTrans = trellisDict.stateTrans;

    % Initialize
    for j = 1:L
        for i = 1:nStates
            viterbi.nextState(i,j).val = [];
        end
    end
    viterbi.PM = 1e9 .* ones(nStates, dataLength(2)+1); %
Initialize as a bignumber so that it will be replaced
    viterbi.nextState(1,1).val =
stateTrans(stateTrans(:,1)==0, 2); % start from state zero
and propagate forward
    viterbi.PM(1,1) = 0;

    % propagate from one state to next
    for time = 1:dataLength(2)
        for state = 0:nStates-1
            % Find connection of current state to next
states
            % after index L, we get next state from Lth
index since it will be same everywhere afterwards
            prev2cur = viterbi.nextState(state+1, (time *
(time < L)) + ( L * (time >= L))).val;
            if ~isempty(prev2cur)
                % Find corresponding output code for a
specific state
                % transition and calculate path metric
                trans1 = stateTrans == [state prev2cur(1)];
trans1 = trans1(:, 1) .* trans1(:, 2);
                trans2 = stateTrans == [state prev2cur(2)];
trans2 = trans2(:, 1) .* trans2(:, 2);
                if strcmp(type, 'HDD')
                    out0 = trellisDict.outputs(:,trans1 ==
1);
                    out1 = trellisDict.outputs(:,trans2 ==
1);
                    PM1 = sum(rx(:, time) ~= out0);
                    PM2 = sum(rx(:, time) ~= out1);
```

```
                elseif strcmp(type, 'SDD')
                    out0 = trellisDict.outputsSDD(:,trans1
== 1);
                    out1 = trellisDict.outputsSDD(:,trans2
== 1);
                    % Calculate negative (instead of
positive) correlation so that PM is minimized in
consistency with HDD
                    PM1 = - out0' * rx(:, time);
                    PM2 = - out1' * rx(:, time);
                else
                    error('Enter valid decoding type: HDD
or SDD')
                end
                % update path metrics and add connections
if suitable path
                if viterbi.PM(prev2cur(1)+1, time+1) >
viterbi.PM(state+1, time)+ PM1
                    viterbi.PM(prev2cur(1)+1, time+1) =
viterbi.PM(state+1, time)+ PM1;
                    viterbi.prevState(prev2cur(1)+1,
time+1) = state;
                end
                if viterbi.PM(prev2cur(2)+1, time+1) >
viterbi.PM(state+1, time)+ PM2
                    viterbi.PM(prev2cur(2)+1, time+1) =
viterbi.PM(state+1, time)+ PM2;
                    viterbi.prevState(prev2cur(2)+1,
time+1) = state;
                end
                % find state connections
                if time < L
                    viterbi.nextState(prev2cur(1)+1,
time+1).val = stateTrans(stateTrans(:,1) == prev2cur(1),
2);
                    viterbi.nextState(prev2cur(2)+1,
time+1).val = stateTrans(stateTrans(:,1) == prev2cur(2),
2);
                end
            end
        end
    end
```

```
    decodedBits = ViterbiBackward(viterbi, trellisDict,
dataLength(2));
end


function decodedBits = ViterbiBackward(viterbi,trellisDict,
dataLength)
    % Traverse along the path with lowest metric
    decodedBits = zeros(dataLength,1);
    for i = dataLength+1:-1:2
        [~, newStateIdx] = min(viterbi.PM(:,i));
        newState = newStateIdx -1;
        oldState = viterbi.prevState(newStateIdx, i);
        trans = trellisDict.stateTrans == [oldState
newState];
        trans = trans(:, 1) .* trans(:, 2);
        decodedBit = trellisDict.inputs(trans==1);
        decodedBits(i-1) = decodedBit;
    end
end


function rxq = UniformQuantizer(rx, N)
    % This function is used from my telecom I course
project 1
    a1 = -1;
    an1 = 1;
    ai = a1 + (2/N)*([1:N+1]-an1);
    h = [0.5, 0.5];
    xi = conv(ai, h, 'same'); % moving average filter
    xi = xi(1:N);
    rxq = zeros(size(rx));
    rxq(rx < ai(1)) = xi(1);
    for i = 1:length(xi)
        rxq((rx < ai(i+1))&(rx >= ai(i))) = xi(i);
    end
    rxq((rx >= ai(N+1))) = xi(N);
end


% SNR in db to N0
function N0 = snr2n0(SNR_db)
    SNR = 10.^(SNR_db./10);
```

```
    N0  = 1 ./ SNR;
end


J = 1;
syms x1 x2 x3 x4 x5 x6 x7 x0 x00 x10 D N
eqn1 = x1 == (x3 * (D^2)*J) + (x2*D*J);
eqn2 = x2 == (x4 * (D^2)*J) + (x5*D*J);
eqn3 = x3 == (x6*D*J) + (x7*(D^2)*J);
eqn4 = x4 == (x0*(D^3)*J*N)+ x1*J*N;
eqn5 = x5 == (x2*(D^2)*J*N) + (x3*D*J*N);
eqn6 = x6 == (x4*D*J*N) + (x5*(D^2)*J*N);
eqn7 = x7 == (x6*(D^2)*J*N) + (x7*D*J*N);
eqn8 = x00 == (x1*(D^3)*J);
eqn9 = x10 == x00 / x0;

sol = solve(eqn1, eqn2, eqn3, eqn4, eqn5, eqn6, eqn7, eqn8,
eqn9);
T = sol.x10;


% The code is dynamic for any constraint length L which
depends on the length
% of generator vectors. Currently implemented for k = 1
%% Non-Catastrophic code

% parity bit vectors
g1 = [1 1 0 1]; % 13
g2 = [1 0 0 1]; % 9
g3 = [1 1 1 1]; % 15
L  = length( g1);

% Create state array and outputs for all possible inputs
trellisDict = trellisDictionary(L, g1, g2, g3);
% trellisDict.outputs(:, 2^L) % Output of Self loop

%% Implement Viterbi decoder assuming perfect receiver and
channel

% nBits = 1000;
% codeword = genRandomCodeword(nBits, L);
% tx = encodeConvolutional(codeword, g1, g2, g3);
% rx = tx;
% decodedBits = ViterbiDecoder(rx, trellisDict, L, 'HDD');
```

```
% bitErrorRate = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);

%% Simulate a binary symmetric channel
trials = 50;
nBits  = 1000;
p = [0, 0.001, 0.002, 0.005, 0.008 0.01:0.01:0.1,
0.15:0.05:0.5];
bitErrorRate = zeros(length(p), 1);
for i = 1:length(p)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        rx = bsc(tx, p(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end

ErrorRates.bscHDD = bitErrorRate;

figure()
loglog(p, ErrorRates.bscHDD);
xlabel('p')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN uncoded

trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = codeword;
```

```
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        rx = rx > 0;
        decodedBits = rx;
        error(j) = 100 * sum(codeword ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNUncoded = bitErrorRate;
figure()
semilogy(SNR_db, ErrorRates.AWGNUncoded);
xlabel('Eb/N0')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN channel with HDD
trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        rx = rx > 0;
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNHDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNHDD);
xlabel('Eb/N0')
ylabel('Bit error Rate')
```

```
%% Simulate transmission over AWGN channel with SDD
trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'SDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNSDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNSDD);
xlabel('Eb/No')
ylabel('Bit error Rate')

%% Simulate transmission over AWGN channel using
quantization with SDD

trials = 50;
nBits  = 1000;
N = [4 16];
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 2);
for k = 1:length(N)
    for i = 1:length(bitErrorRate)
        disp(i)
        error = zeros(trials,1);
        for j = 1:trials
            codeword = genRandomCodeword(nBits, L);
```

```
            tx = encodeConvolutional(codeword, g1, g2, g3);
            tx(tx==0) = -1;
            rx = AWGN(tx, N0(i));
            rx = UniformQuantizer(rx, N(k));
            decodedBits = ViterbiDecoder(rx, trellisDict,
L, 'SDD');
            error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
        end
        bitErrorRate(i, k) = mean(error);
    end
end
ErrorRates.AWGNQSDD = bitErrorRate;

figure()
semilogy(SNR_db, ErrorRates.AWGNQSDD(:,1));
hold on
semilogy(SNR_db, ErrorRates.AWGNQSDD(:,2));
legend('N = 4', 'N = 16');
xlabel('Eb/No')
ylabel('Bit error Rate')
%% Catastrophic code

% parity bit vectors
g1 = [1 0 1 0]; %10
g2 = [1 1 0 0]; %12
g3 = [1 0 0 1]; %9
L  = length( g1);

% Create state array and outputs for all possible inputs
trellisDict = trellisDictionary(L, g1, g2, g3);
% trellisDict.outputs(:,16)

%% Simulation  of catastrophic code over BSC

trials = 50;
nBits  = 1000;
p = [0, 0.001, 0.002, 0.005, 0.008 0.01:0.01:0.1,
0.15:0.05:0.5];
bitErrorRate = zeros(length(p), 1);
for i = 1:length(p)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
```

```
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        rx = bsc(tx, p(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'HDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.bscCatHDD = bitErrorRate;

figure()
loglog(p, ErrorRates.bscCatHDD);
hold on
loglog(p, ErrorRates.bscHDD);
legend('Catastrophic', 'Normal')
xlabel('p')
ylabel('Bit error Rate')

%% Simulation  of catastrophic code over AWGN with SDD

trials = 50;
nBits  = 1000;
SNR_db = [-10:2:-2, -1:10];
N0 = snr2n0(SNR_db);
bitErrorRate = zeros(length(N0), 1);
for i = 1:length(bitErrorRate)
    disp(i)
    error = zeros(trials,1);
    for j = 1:trials
        codeword = genRandomCodeword(nBits, L);
        tx = encodeConvolutional(codeword, g1, g2, g3);
        tx(tx==0) = -1;
        rx = AWGN(tx, N0(i));
        decodedBits = ViterbiDecoder(rx, trellisDict, L,
'SDD');
        error(j) = 100 * sum(codeword' ~=
decodedBits)/length(decodedBits);
    end
    bitErrorRate(i) = mean(error);
end
ErrorRates.AWGNCatSDD = bitErrorRate;
```

```
figure()
semilogy(SNR_db, ErrorRates.AWGNCatSDD);
hold on
semilogy(SNR_db, ErrorRates.AWGNSDD);
legend('Catastrophic', 'Normal')
xlabel('Eb/No')
ylabel('Bit error Rate')
```