CT216 Convolution Code Report

G 34

April 2025

Contents

[1] Codes for used Functions are given in *Section 6 - Appendix*.

1 What is Transfer Function and How to Find It

1.1 Analogy

Like an LTI system where we have some input which can be represented as a impulse train, and find a impulse response to that LTI system. Then we can convolve input with impulse response to get output. In time domain convolution is a multiplication in frequency domain by a transfer function.

In time Domain

$$y(t) = x(t) * h(t)$$

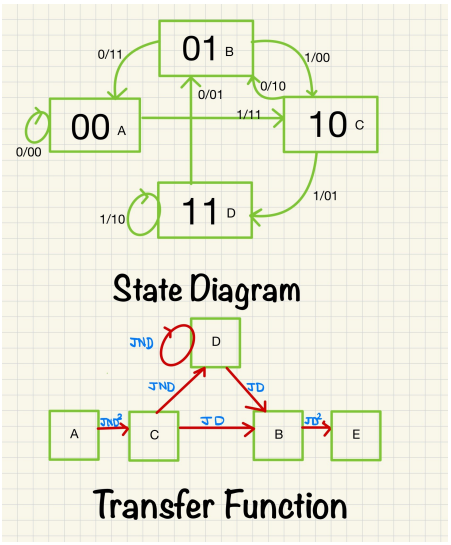
In Frequency Domain

$$Y(w) = X(w)H(w)$$

Similarly in Convolutional Codes time domain impulse response can be found by generator matrix passing a $[1, 0, 0, 0, ..., 0]$ as impulse and then recording it's output.This is it's impulse response. Then we can find it's output by just delay and linear combination of impulse response.
We can see it from a different perspective. Just like time and frequency we can have time and hamming distance as 2 sides of the same encoder.

1.2 Method of finding a Transfer Function for rate $1/2$ $K_c = 3$

1.2.1 State Diagram



Here Exponent of J is a counter for path length.
Also Exponent of N represent's number of 1^s in the codeword.
And Exponent of D represents hamming distance for a codeword.

1.2.2 State Function

$$X_c = JND^2X_a$$
$$X_d = JND(X_c + X_d)$$
$$X_b = JD(X_c + X_d)$$
$$X_e = JD^2X_b$$

1.2.3 Transfer Function

$$\frac{X_e}{X_a} = \frac{J^3ND^5}{1 - JND}$$
$$T(J, N, D) = \sum_{i=0}^{\infty} J^{i+3}N^{i+1}D^{i+5}$$
$$d_{free} = 5$$
$$T(J, N, D) = \sum_{d=d_{free}}^{\infty} J^{d-2}N^{d-4}D^d$$

2 Derivation of Soft Decoding Parameters

2.1 Branch Metric

$$\mu_j^i = \sum_{m=1}^{n} r_{jm}(2C_{jm}^i - 1)$$

$$r_{jm} = \sqrt{e_c}(2C_{jm} - 1) + n_{jm}$$

$$n_{jm} -> N(0, \sigma^2)$$

Thus r_{jm} follows the following distribution

$$r_{jm} -> N(\sqrt{e_c}(2C_{jm} - 1), \sigma^2)$$

Here, $\sigma^2 = \frac{N_o}{2}$

2.2 Correlation Metric

$$CM^i = \sum_{j=1}^{N} \mu_j^i$$

2.3 Finding Probability of First Error at distance d

Since Convolution Codes are LTI system. Thus all zeros codeword is a valid codeword. We will use this property in all the analysis.

$$P_{firstError}^d = P(CM^1 > CM^0)$$

Since we have transmitted an all zero codeword so its correlation metric would be lower compared to path with 'd' hamming distance from the sent codeword.

$$CM^1 > CM^0$$

$$P(CM^1 - CM^0 > 0) = P(\sum_{j=1}^{N}(\mu_j^1 - \mu_j^0))$$

$$P(CM^1 - CM^0 > 0) = P(2\sum_{j=1}^{N}\sum_{m=1}^{n} r_{jm}(C_{jm}^1 - C_{jm}^0) > 0)$$

Since we have sequences differing at 'd' position's thus all other terms in the sum will evaluate to 0.

$$P(CM^1 - CM^0 > 0) = P(\sum_{l=1}^{d} r_l > 0)$$

Also r_l is a normal random variable. Thus their sum is also a normal random variable.

$$\sum_{l=1}^{d} r_l -> N(\sqrt{e_c}\sum_{l=1}^{d}(2C_l - 1), d\sigma^2)$$

Since input was all zero codeword, thus $C_l = 0$.

$$X = \sum_{l=1}^{d} r_l -> N(-d\sqrt{e_c}, d\sigma^2)$$

$$Y = \frac{X + d\sqrt{e_c}}{\sigma\sqrt{d}}$$

For $X > 0$ it implies that $Y > \sqrt{\frac{de_c}{\sigma^2}}$. Thus $P(X > 0) = P(Y > \sqrt{\frac{2e_c d}{N_0}}) = Q(\sqrt{\frac{2e_c d}{N_0}})$

$$P^d_{firstError} = Q(\sqrt{\frac{2e_c d}{N_0}})$$

For a approximation we can write Q function with upper bound as follows:

$$P^d_{firstError} = Q(\sqrt{\frac{2e_c d}{N_0}}) < e^{-\frac{e_c d}{N_0}}$$

2.4 Finding Probability of First error irrespective of length

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} (number\, of\, possible\, path\, with\, hamming\, distance\, d) * P^d_{firstError}$$

This is just considering all possible paths of distance 'd' where d is from $d = d_{free}$ till $d = \infty$.

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} a_d * P^d_{firstError}$$

Here a_d is the number of paths with hamming distance 'd' from actual codeword. We can write Probablity in another way as follows.

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} a_d (e^{-\frac{e_c}{n_0}})^d$$

Here we can write $e^{-\frac{e_c}{n_0}}$ as D just as in transfer function. Also since a_d can be found from transfer function we can write Probablity in terms of transfer function.

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} a_d (D)^d = T(J, N, D)|J = 1, N = 1, D = e^{-\frac{e_c}{N_0}}$$

2.5 Bit Error Rate

$$BER^d = a_d(Number\,of\,mismatch\,bits)P^d_{firstError}$$

Thus over all path length's it will be the following:

$$BER < \sum_{d=d_{free}}^{\infty} a_d(Number\,of\,mismatch\,bits)P^d_{firstError}$$

We know that in transfer function exponent of N represent's number of 1's in the sequence. Since we are transmitting all zero codeword which is also the input. Thus there will be exponent of N number of mismatch at any distance 'd'. Thus if we want to include transfer function then we can write it as follows:

$$BER = \sum_{d=d_{free}}^{\infty} a_d g(d) D^d$$

This form can be acheived by keeping $D = e^{-\frac{e_c}{N_0}}$ and $J = 1, N = 1$

$$BER < \frac{d(T(J,N,D))}{dN} | J = 1, N = 1, D = e^{-\frac{e_c}{N_0}}$$

2.5.1 BER for our example

In our example Transfer Function is as follows:

$$T(J,N,D) = \sum_{d=d_{free}}^{\infty} J^{d-2} N^{d-4} D^d$$

Thus BER is calculated as follows:

$$BER < \frac{d(T(J,N,D))}{dN} | J = 1, N = 1, D = e^{-\frac{e_c}{N_0}}$$

$$\frac{dT}{dN} = \sum_{d=5}^{\infty} J^{d-2}(d-4) N^{d-5} D^d$$

$$BER < \sum_{d=5}^{\infty} (d-4) D^d$$

This is a arithmetico-Geometric Progression. Thus,

$$BER = \frac{D^5}{(1-D)^2}$$

Here $D = e^{-\frac{e_c}{N_0}} = e^{-rate\frac{e_b}{N_0}}$

$$BER = \frac{(e^{-rate\frac{e_b}{N_0}})^5}{(1 - e^{-rate\frac{e_b}{N_0}})^2}$$

3 Derivation of Hard Decoding Parameters

3.1 Probability of first error event for length d

Since in hard decision decoding there is hamming distance as a comparison between 2 codeword sequences.
If a sequence with hamming distance d is received with error's less than $\frac{d+1}{2}$ then it is solvable. Else not solvable.

$$P^d_{firstError} = \sum_{k=\frac{d+1}{2}}^{d} \binom{d}{k} (P_{bitFlip})^k (1 - P_{bitFlip})^{d-k}$$

This can be approximated by the below formula:

$$P^d_{firstError} = \sqrt{4P_{bitFlip}(1 - P_{bitFlip})}$$

$$P_{bitFlip} = Q(\sqrt{\frac{2e_c}{N_0}})$$

3.2 Probability of first error irrespective of d

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} a_d P^d_{firstError}$$

$$P_{firstError} < \sum_{d=d_{free}}^{\infty} a_d (\sqrt{4P_{bitFlip}(1 - P_{bitFlip})})^d$$

Here we can write $D = \sqrt{4P_{bitFlip}(1 - P_{bitFlip})}$

$$P_{firstError} < T(J, N, D)|J = 1, N = 1, D = \sqrt{4P_{bitFlip}(1 - P_{bitFlip})}$$

3.3 Bit Error Rate

$$BER^d = a_d(Number\,of\,mismatch\,bits)P^d_{firstError}$$

We know that in transfer function exponent of N represent's number of 1's in the sequence. Since we are transmitting all zero codeword which is also the input. Thus there will be exponent of N number of mismatch at any distance 'd'. Thus if we want to include transfer function then we can write it as follows:

$$BER < \frac{d(T(J, N, D))}{dN}|J = 1, N = 1, D = \sqrt{4P_{bitFlip}(1 - P_{bitFlip})}$$

3.3.1 BER for our example

In our example Transfer Function is as follows:

$$T(J,N,D) = \sum_{d=d_{free}}^{\infty} J^{d-2}N^{d-4}D^d$$

Thus BER is calculated as follows:

$$BER < \frac{d(T(J,N,D))}{dN}|J=1,N=1,D=\sqrt{4P_{bitFlip}(1-P_{bitFlip})}$$

$$\frac{dT}{dN} = \sum_{d=5}^{\infty} J^{d-2}(d-4)N^{d-5}D^d$$

$$BER < \sum_{d=5}^{\infty}(d-4)D^d$$

This is a arithmetico-Geometric Progression. Thus,

$$BER = \frac{D^5}{(1-D)^2}$$

Here $D = \sqrt{4P_{bitFlip}(1-P_{bitFlip})}$
Also Probablity of bit flip is:

$$P_{bitFlip} = Q(\sqrt{\frac{2e_c}{N_0}})$$

4 Convolution Code and results

4.1 Main Simulation Running Code:[2]

Part 1) This is Code for Running Convolution Code for a single input sequence with Hard Decoding:

```
noOfInput=1;
lenOfInput=8;
noOfState=8;
lenOfOutput=2;
maxV=1;
rate=noOfInput/lenOfOutput;
sLog=linspace(4,7,7);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=2*rate.*s;% value of Es/No in normal scale
Diagram=stateDiagram(noOfState,noOfInput);
Pow=4;
S=1/sqrt(g(Pow));
input=randi([0,1],1,lenOfInput-log2(noOfState));
```

[2]Codes for used Functions are given in *Section 6 - Appendix.*

```
zeroPadd=zeros(1,log2(noOfState));
input=[input,zeroPadd]
output=[];
prevState=zeros(1,log2(noOfState));
for i=1:lenOfInput/noOfInput
    [out,prevState]=Encoder(input((i-1)*noOfInput+1:i*noOfInput),
        prevState);
    output=[output,out];
end
output
send=BPSKSender(output,maxV);
send=AWGNChannel(send,S);
receive=BPSKHardReceiver(send);
decoded=decoderHard(receive,lenOfOutput,noOfState,Diagram)
BER=sum(xor(input,decoded))/lenOfInput
```

Part 2) This is Code for Running Convolution Code for a single input sequence with Soft Decoding:

```
noOfInput=1;
lenOfInput=8;
noOfState=8;
lenOfOutput=2;
maxV=1;
noOfBand=8;
rate=noOfInput/lenOfOutput;
sLog=linspace(4,7,7);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=2*rate.*s;% value of Es/No in normal scale
Diagram=stateDiagram(noOfState,noOfInput);
Pow=4;
S=1/sqrt(g(Pow));
input=randi([0,1],1,lenOfInput-log2(noOfState));
zeroPadd=zeros(1,log2(noOfState));
input=[input,zeroPadd]
output=[];
prevState=zeros(1,log2(noOfState));
for i=1:lenOfInput/noOfInput
    [out,prevState]=Encoder(input((i-1)*noOfInput+1:i*noOfInput),
        prevState);
    output=[output,out];
end
output
send=BPSKSender(output,maxV);
send=AWGNChannel(send,S);
comp=zeros(size(send));
receive=BPSKSoftReceiver(send,noOfBand,maxV,S);
decoded=decoderSoft(receive,lenOfOutput,noOfState,Diagram,noOfBand
    ,1)% 1 means Euclid Distance 0 means manhattan Distance
BER=sum(xor(input,decoded))/lenOfInput
```

Part 3) This is Code for running Monte Carlo Simulation on Hard and Soft Convolution Code.
Output is a Graph for BER(Bit Error Rate) and PER (Probablity of error in decoding)
Input is the energy of noise, size of Encoder and length of input:

```
noOfInput=1;%To select the size of encoder
lenOfInput=50;
noOfState=8;%To select the size of encoder
lenOfOutput=2;
maxV=1;
noOfBand=8;
rate=noOfInput/lenOfOutput;
sLog=linspace(0,10,11);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=s;% value of Es/No in normal scale
Diagram=stateDiagram(noOfState,noOfInput);
nMax=10000;
BERS=zeros(1,11);
BERH=zeros(1,11);
PerrDecSoft=zeros(1,11);
PerrDecHard=zeros(1,11);
inputSet=zeros(nMax,lenOfInput-log2(noOfState));
for i=1:nMax
    inputSet(i,:)=randi([0,1],1,lenOfInput-log2(noOfState));
end
for Pow=1:11
    S=1/sqrt(g(Pow));
    for iter=1:nMax
        input=inputSet(iter,:);
        zeroPadd=zeros(1,log2(noOfState));
        input=[input,zeroPadd];
        output=[];
        prevState=zeros(1,log2(noOfState));
        for i=1:lenOfInput/noOfInput
            [out,prevState]=Encoder(input((i-1)*noOfInput+1:i*
                noOfInput),prevState);
            output=[output,out];
        end
        send=BPSKSender(output,maxV);
        send=AWGNChannel(send,S);
        receive=BPSKSoftReceiver(send,noOfBand,maxV,S);
        receive1=BPSKHardReceiver(send);
        decoded=decoderSoft(receive,lenOfOutput,noOfState,Diagram,
            noOfBand,1);% Euclid Distance
        decoded1=decoderHard(receive1,lenOfOutput,noOfState,Diagram
            );
        BERS(Pow)=BERS(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decoded(1:end-log2(noOfState))));
        BERH(Pow)=BERH(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decoded1(1:end-log2(noOfState))));
        % We dont need to know what the last 3 bits are since they
            will be 0
        if(~isequal(input(1:end-log2(noOfState)),decoded(1:end-log2
            (noOfState))))
            PerrDecSoft(Pow)=PerrDecSoft(Pow)+1;
        end
```

```
        if(~isequal(input(1:end-log2(noOfState)),decoded1(1:end-
            log2(noOfState))))
            PerrDecHard(Pow)=PerrDecHard(Pow)+1;
        end
    end
end
BERS=BERS/(nMax*length(input));
BERH=BERH/(nMax*length(input));
PerrDecSoft=PerrDecSoft/nMax;
PerrDecHard=PerrDecHard/nMax;
semilogy(sLog,BERS);
hold on;
semilogy(sLog,BERH);
hold off;
semilogy(sLog,PerrDecSoft);
hold on;
semilogy(sLog,PerrDecHard);
hold off;
```

Part 4) Below is the output for interweaved data bits v/s normal data bits This is done in order to reduce chances of burst error corrupting a large chunk of information

It is done so for rate 1/2 code by first sending all 1st output bit of each input and then sending 2nd output bit.

This way if some 1st bits gets corrupted then we can extract some information from remaining 2nd output bits since they were not in series otherwise both would have gotten corrupted:

```
% Now we see for interweaving the data and then sending it.
noOfInput=1;
lenOfInput=50;
noOfState=8;
lenOfOutput=2;
maxV=1;
noOfBand=8;
rate=noOfInput/lenOfOutput;
sLog=linspace(0,10,11);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=s;% value of Es/No in normal scale is 2eb/
Diagram=stateDiagram(noOfState,noOfInput);
nMax=3000;
BERSI=zeros(1,11);% Interweaved data BER for soft decesion decoding
BERSN=zeros(1,11);% Noraml data BER for soft decesion decoding
inputSet=zeros(nMax,lenOfInput-log2(noOfState));
for i=1:nMax
    inputSet(i,:)=randi([0,1],1,lenOfInput-log2(noOfState));
end
for Pow=1:11
    S=1/sqrt(g(Pow));
    for iter=1:nMax
        input=inputSet(iter,:);
        zeroPadd=zeros(1,log2(noOfState));
        input=[input,zeroPadd];
```

```
        output=[];
        prevState=zeros(1,log2(noOfState));
        for i=1:lenOfInput/noOfInput
            [out,prevState]=Encoder(input((i-1)*noOfInput+1:i*
                noOfInput),prevState);
            output=[output,out];
        end
        % Here we will try to seperate output bits and send them
            after some
        % delay
        O1=output(1:2:end);
        O2=output(2:2:end);
        outputI=[O1,O2];
        sendI=BPSKSender(outputI,maxV);
        sendN=BPSKSender(output,maxV);
        sendI=BurstErrorChannel(sendI,S);
        sendN=BurstErrorChannel(sendN,S);
        receiveI=BPSKSoftReceiver(sendI,noOfBand,maxV,S);
        receiveN=BPSKSoftReceiver(sendN,noOfBand,maxV,S);
        for i=1:length(outputI)
            receiveM(i)=mod(i,2)*receiveI(int8(i/2))+mod(i+1,2)*
                receiveI(int8(i/2)+lenOfInput);
        end
        decodedI=decoderSoft(receiveM,lenOfOutput,noOfState,Diagram
            ,noOfBand,1);
        decodedN=decoderSoft(receiveN,lenOfOutput,noOfState,Diagram
            ,noOfBand,1);
        BERSI(Pow)=BERSI(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decodedI(1:end-log2(noOfState))));
        BERSN(Pow)=BERSN(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decodedN(1:end-log2(noOfState))));
        % We dont need to know what the last 3 bits are since they
            will be 0
    end
end
BERSI=BERSI/(nMax*length(input));
BERSN=BERSN/(nMax*length(input));
semilogy(sLog,BERSI);
hold on;
semilogy(sLog,BERSN);
hold off;
```

Part 5) Now we see in Soft decesion rather than Euclidean distance when we use manhattan distance the BER performance we get:

```
noOfInput=1;
lenOfInput=10;
noOfState=8;
lenOfOutput=2;
maxV=1;
noOfBand=8;
rate=noOfInput/lenOfOutput;
sLog=linspace(0,10,11);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=s;% value of Es/No in normal scale is 2eb/
```

```
Diagram=stateDiagram(noOfState,noOfInput);
nMax=3000;
BERSE=zeros(1,11);% Interweaved data BER for soft decesion decoding
BERSM=zeros(1,11);% Noraml data BER for soft decesion decoding
inputSet=zeros(nMax,lenOfInput-log2(noOfState));
for i=1:nMax
    inputSet(i,:)=randi([0,1],1,lenOfInput-log2(noOfState));
end
for Pow=1:11
    S=1/sqrt(g(Pow));
    for iter=1:nMax
        input=inputSet(iter,:);
        zeroPadd=zeros(1,log2(noOfState));
        input=[input,zeroPadd];
        output=[];
        prevState=zeros(1,log2(noOfState));
        for i=1:lenOfInput/noOfInput
            [out,prevState]=Encoder(input((i-1)*noOfInput+1:i*
                noOfInput),prevState);
            output=[output,out];
        end
        send=BPSKSender(output,maxV);
        send=AWGNChannel(send,S);
        receive=BPSKSoftReceiver(send,noOfBand,maxV,S);
        decodedE=decoderSoft(receive,lenOfOutput,noOfState,Diagram,
            noOfBand,1);% 1 means Euclid Distance
        decodedM=decoderSoft(receive,lenOfOutput,noOfState,Diagram,
            noOfBand,0);% 0 means Manhattan Distance
        BERSE(Pow)=BERSE(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decodedE(1:end-log2(noOfState))));
        BERSM(Pow)=BERSM(Pow)+sum(xor(input(1:end-log2(noOfState)),
            decodedM(1:end-log2(noOfState))));
        % We dont need to know what the last 3 bits are since they
            will be 0
    end
end
BERSE=BERSE/(nMax*length(input));
BERSM=BERSM/(nMax*length(input));
semilogy(sLog,BERSE);
hold on;
semilogy(sLog,BERSM);
hold off;
```

Part 6) Analytical Approximation of BER soft and hard to the Kc=3 and 1 input 2 output encoder rate 1/2 convolution code:

```
sLog=linspace(0,10,11);% value of Eb/N0 in db
s=10.^(sLog/10);% value of Eb/N0 in normal scale
g=s;% value of Es/No in normal scale
u=exp(-0.5*s);
BERS=zeros(1,11);
for i=1:11
    BERS(i)=u(i)^5/(1-u(i))^2;
end
semilogy(sLog,BERS);
```

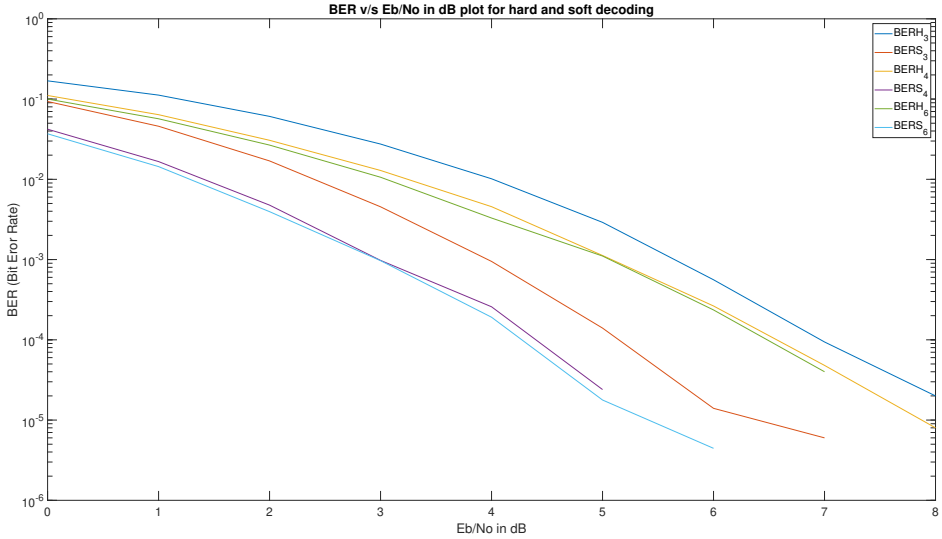
```
hold on;

% Creating data points for q function to be evaluated at using
    matQfunc()
% since qFunc is in communications toolbox but we are not allwed to
     use it.
q=zeros(1,100000);
x=linspace(-1000,1000,100000);% Sampling points to later do a
    linear interpolation to get proper value.
for i=1:100000
    q(i)=matQfunc(x(i));
end

P=sqrt(2.*s);
PBF=interp1(x,q,P);
PBER=zeros(1,11);
D=zeros(1,11);
for i=1:11
    D(i)=sqrt(8*PBF(i)*(1-PBF(i)));
    PBER(i)=D(i)^5/(1-D(i))^2;
end
semilogy(sLog,PBER);
hold off;


function val=matQfunc(min)
N=100000;
nMax=10000;
dx=(nMax-min)/N;
x=linspace(min,nMax,N);
y=(1/sqrt(2*pi)).*exp(-(x.^2)/2);
Ay=sum(y);
val=Ay*dx;
end
```

4.2 BER V/S Eb/No, Output for Kc=3,4,6

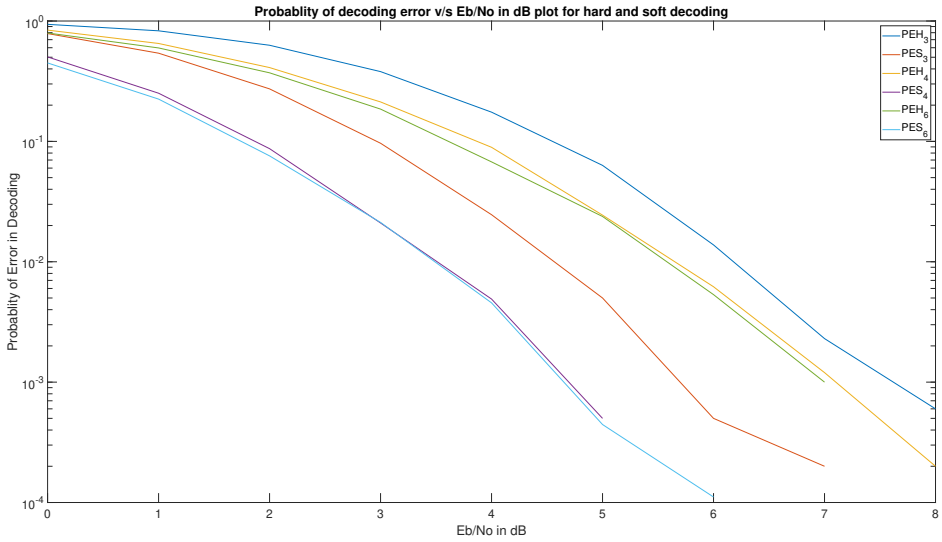


The plots depict the performance of the Viterbi hard and soft decoders for different rates and constraint lengths.[3]

- From the above graph we see a general trend of Bit Error Rate decreasing as the Noise decreases.
- We can see that for a three state system the difference between Hard and Soft Decoding is around 2db.
- Thus, when noise is high but we want to limit time complexity, we can substitute our three state soft decoder for a Sixteen state Hard Decoder, since Hard Decoder has less Time Complexity.
- So, in order to maximize the bit rate of our system in varying noise scenario, we can hop from one type of encoder to another to maximize bit rate and minimize error

[3]The data used for plotting all the graphs is included in the ZIP file named "Convolution_Code_Plot_Data" under the Folder named "Convolution_Code_34"

4.3 Probability of decoding error v/s Eb/No,Output for Kc =3,4,6



Performance of both Viterbi hard and soft for each case increases with the increase in SNR.The BER and PDE is higher for lower SNR which suggests that Viterbi decoding performs better for minimal noise. Larger constraint length enhances error correction significantly but the cost is significant increase in time due to increase in computation complexity.The overall performance of soft decision decoding is much better than the hard decision. The reasons can be:
i) In hard decision decoding, each of the received symbol is demodulated to bits considering a specific threshold. Because of this information about how reliable a bit is lost.
ii) Soft decision in addition to representing the bit received it also attaches the information of how reliable the bit is. Thus reducing errors due to misjudged bit values.

5 Comparison of theoretical v/s simulation

- For Soft Decoding:

$$BER < \frac{d(T(J,N,D))}{dN}|J=1,N=1,D=e^{-\frac{e_c}{N_0}}$$

$$BER = \frac{(e^{-rate\frac{e_b}{N_0}})^5}{(1-e^{-rate\frac{e_b}{N_0}})^2}$$

- For Hard Decoding:

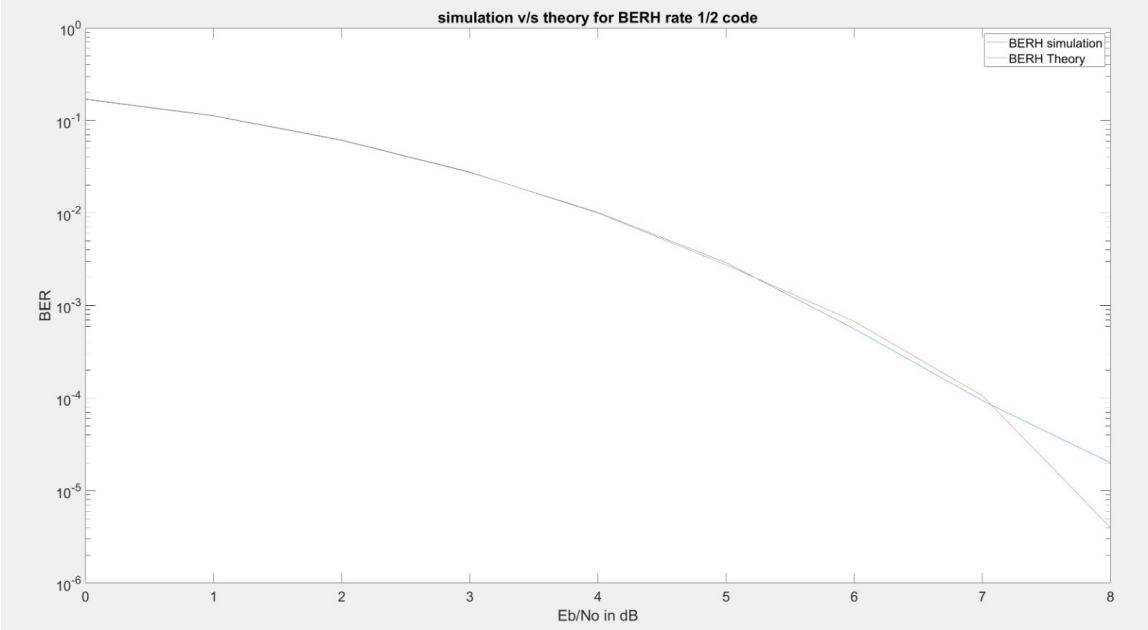
$$BER < \frac{d(T(J,N,D))}{dN}|J=1,N=1,D=\sqrt{4P_{bitFlip}(1-P_{bitFlip})}$$

$$BER = \frac{D^5}{(1-D)^2}$$

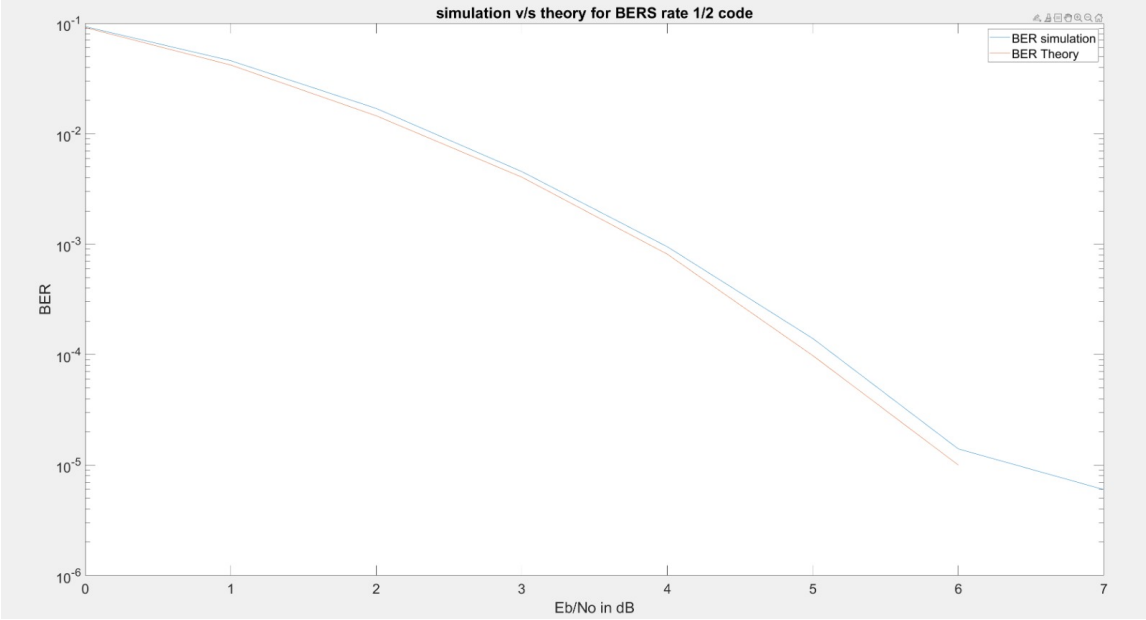
Here $D = \sqrt{4P_{bitFlip}(1-P_{bitFlip})}$
Also Probablity of bit flip is:

$$P_{bitFlip} = Q(\sqrt{\frac{2e_c}{N_0}})$$

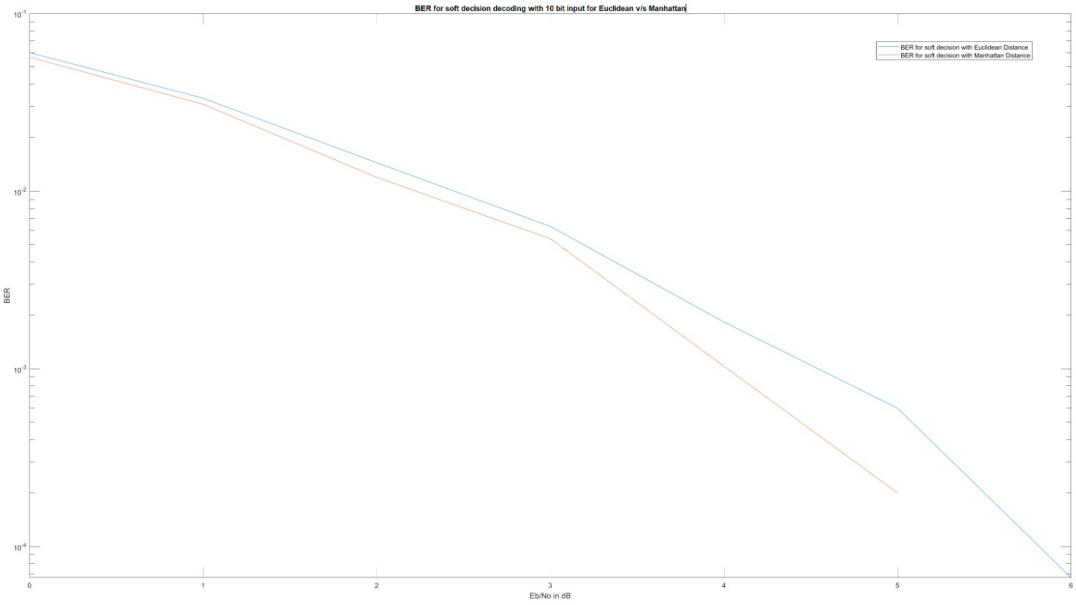
5.1 For Hard decoding :



5.2 For soft decoding :



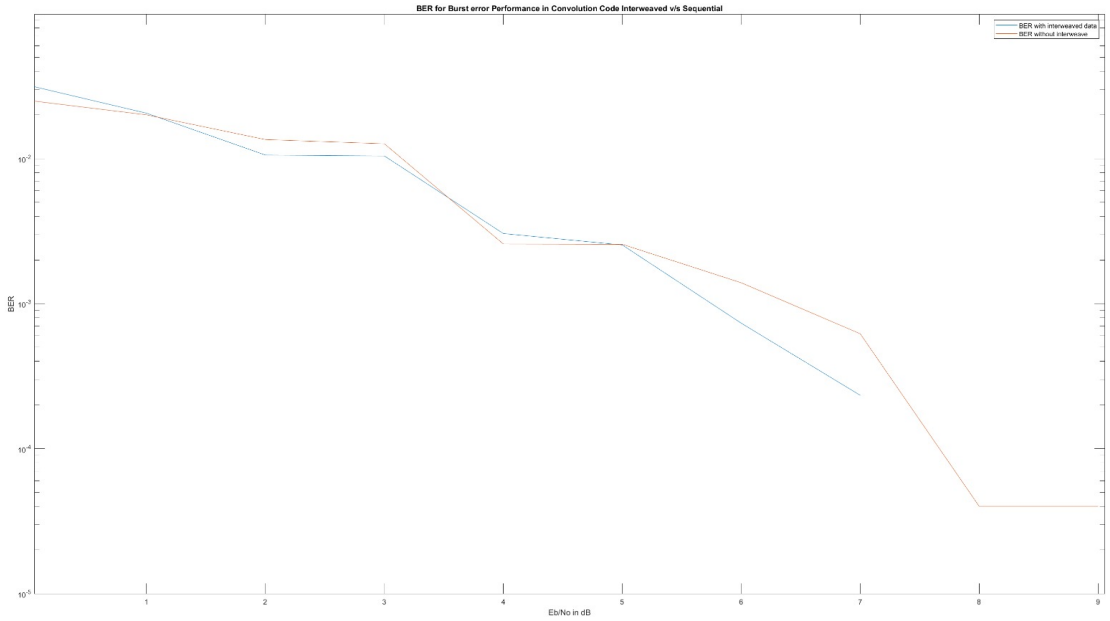
5.3 Euclidean v/s Manhattan Distance comparison



As we can see, Manhattan distance (for small input) performs very well compared to

euclidean distance as it takes the absolute difference between the demodulated bit and the original value of 0[0] or 7[1]. So, we can send data in small packets through the Manhattan distance branch metric.

5.4 Interweaved v/s Sequential Data Transmission for Burst Error Channel



Motivation: Since error in channel for space communications occurs in burst. This may corrupt the whole output length. So instead we send all c_0's first then c_1's after it. Thus reducing probability of all corrupt codeword.

6 Appendix

6.1 Source Code for Functions

6.1.1 Encoder

```
function [output,newState]=Encoder(input,prevState)
[~,lenOfInput]=size(input);
newState=[input,prevState];
newState(end-lenOfInput+1:end)=[];
% Output Algorithm
% % for rate 1/2 kc=3 output:
G=[1,0;
   1,1;
   1,1];
% For kc=4 rate 1/3 code:
```

```
% G=[1,1,1;
%     1,1,0;
%     1,0,1;
%     1,1,0];
% For kc=6 r=1/3 code:
% G=[1,0,1;
%     1,1,0;
%     1,1,1;
%     1,0,1;
%     1,1,1;
%     1,1,0];
output=newState*G;
output=mod(output,2);
end
```

6.1.2 State Diagram

```
function Diagram=stateDiagram(noOfState,lenOfInput)
Diagram=zeros(2^lenOfInput,noOfState,2);
lenOfState=log2(noOfState);
s1=(2^(lenOfState-1))*(1/2).^(0:lenOfState-1);
s1=transpose(s1);
for input=1:2^lenOfInput
    for state=1:noOfState
        [output,nxtState]=Encoder(flip(bitget(input-1,1:lenOfInput)
            ),flip(bitget(state-1,1:lenOfState)));%de2bi(input-1,
            lenOfInput,"left-msb"),de2bi(state-1,lenOfState,"left-
            msb")
        lenOfOutput=length(output);
        s2=(2^(lenOfOutput-1))*(1/2).^(0:lenOfOutput-1);
        s2=transpose(s2);
        % nxtState=bi2de(nxtState,"left-msb");
        nxtState=nxtState*s1;
        % output=bi2de(output,"left-msb");
        output=output*s2;
        Diagram(input,state,1)=output;
        Diagram(input,state,2)=nxtState;
    end
end
end
```

6.1.3 BPSK Sender

```
function input=BPSKSender(input,maxV)
input(:)=(2*maxV)*input(:)-maxV;
end
```

6.1.4 Channels for Transmission

Part 1) A simple AWGN Channel for transmission.

```
function e=AWGNChannel(seq,stDev)
[~,sSeq]=size(seq);
e=stDev*randn(1,sSeq);
e=seq(:)+e(:);
e=transpose(e);
end
```

Part 2) A Burst Error Channel for imitating transmission in space as errors occur in burst

```
function e=BurstErrorChannel(seq,stDev)
%Burst Error
[~,szOfInput]=size(seq);
PBurstOccurance=0.2;
ValBurst=3*stDev;
lenOfBurst=5;
noOfSector=szOfInput/lenOfBurst;
noOfBurst=uint8(PBurstOccurance*noOfSector);
pos=randi([1,noOfSector],[1,noOfBurst]);
for i=1:noOfBurst
    for j=1:lenOfBurst
        seq(pos(i)+j)=seq(pos(i)+j)+ValBurst;
    end
end
e=seq;
end
```

6.1.5 BPSK Receivers (Hard and Soft)

Part 1) BPSK Hard Receiver

```
function receive=BPSKHardReceiver(input)
[~,noOfSym]=size(input);
receive=[];
for i=1:noOfSym
    if(input(i)>0)
        receive=[receive , 1];
    else
        receive=[receive , 0];
    end
end
end
```

Part 2) BPSK Soft Receiver (does the quantization based on the received voltage to determine the quality of the received bit.)

```
function receive=BPSKSoftReceiver(input,NoOfBand,A,s)
[~,noOfSym]=size(input);
receive=[];
basket=linspace(-A,A,NoOfBand-1);
for i=1:noOfSym
    if(input(i)<=-A)
        receive=[receive,0];
    elseif(input(i)>=A)
        receive=[receive,NoOfBand-1];
    else
        for j=2:NoOfBand-1
            if(input(i)>basket(j-1)&&input(i)<=basket(j))
                receive=[receive,j-1];
                break;
            end
        end
    end
end
end
% LR=exp(input(:).*(2*A/s^2));
% maxL=A;
% basket=linspace(exp(-2*maxL*A/s^2),exp(2*maxL*A/s^2),NoOfBand-1);
% receive=[];
% for i=1:noOfSym
%    if(LR(i)<=basket(1))
%        receive=[receive,0];
%    elseif(LR(i)>=basket(NoOfBand-1))
%        receive=[receive,NoOfBand-1];
%    else
%        for j=2:NoOfBand-1
%            if(LR(i)>basket(j-1)&&LR(i)<=basket(j))
%                receive=[receive,j-1];
%            end
%        end
%    end
% end
% end
```

6.1.6 Viterbi Decoder

Part 1) Viterbi Hard Decoder which works on the sequence generated by the BPSK Hard Receiver

```
function message=decoderHard(receivedOutput,lenOfOutput,noOfState,
    Diagram)

%modifying format of received output for further process
[~,Outsize]=size(receivedOutput);
n=Outsize/lenOfOutput;
receivedOutput=reshape(receivedOutput,lenOfOutput,n);
receivedOutput=transpose(receivedOutput);
s=(2^(lenOfOutput-1))*(1/2).^(0:lenOfOutput-1);
s=transpose(s);
```

```
receivedOutput=receivedOutput*s;
% receivedOutput=bi2de(receivedOutput,"left-msb");
%receivedOutput=flip(receivedOutput);
%now creating a array implementation of trellis tree
weight=zeros(noOfState,n+1,2);
weight(:,:,:)=Inf;
for time=1:n+1
    if(time~=1)
        for state=1:noOfState
            [PrevState,output]=PreviousState(state-1,Diagram);%
                state-1 is actual state
            [~,Size]=size(PrevState);
            wmin=Inf;
            prev=Inf;
            for counter=1:Size
                w=weight(PrevState(counter)+1,time-1,1)+
                    hammingDistance(receivedOutput(time-1),output(
                    counter),lenOfOutput);
                if(w<wmin)
                    wmin=w;
                    prev=PrevState(counter)+1;
                end
            end
            weight(state,time,1)=wmin;
            weight(state,time,2)=prev;%This previous state is
                actualState+1
        end
    else
        weight(1,time,1)=0;
    end
end
% We have now created Trellis tree
pos=n;
message=[];
% [~,indx]=min(weight(:,n+1,1));
indx=1;
while pos>0
    from1=weight(indx,pos+1,2)-1;
    if(from1==Inf)
        from=0;
    else
        from=from1;
    end
    to=indx-1;
    msg=InputForThistransition(from,to,Diagram);
    message=[msg,message];
    indx=from1+1;
    pos=pos-1;
end
end
```

Part 2) Viterbi Soft Decoder which works on the sequence generated by the BPSK Soft Receiver

```
function message=decoderSoft(receivedOutput,lenOfOutput,noOfState,
    Diagram,Band,Method)

%modifying format of received output for further process
[~,Outsize]=size(receivedOutput);
n=Outsize/lenOfOutput;
receivedOutput=reshape(receivedOutput,lenOfOutput,n);
receivedOutput=transpose(receivedOutput);
%now creating a array implementation of trellis tree
weight=zeros(noOfState,n+1,2);
weight(:,:,:)=Inf;
for time=1:n+1
    if(time~=1)
        for state=1:noOfState
            [PrevState,output]=PreviousState(state-1,Diagram);%
                state-1 is actual state
            [~,Size]=size(PrevState);
            wmin=Inf;
            prev=Inf;
            for counter=1:Size
                if(Method)
                    BM=euclidDistance(receivedOutput(time-1,:),
                        output(counter),lenOfOutput,Band);
                else
                    BM=manhattanDistance(receivedOutput(time-1,:),
                        output(counter),lenOfOutput,Band);
                end
                w=weight(PrevState(counter)+1,time-1,1)+BM;
                if(w<wmin)
                    wmin=w;
                    prev=PrevState(counter)+1;
                end
            end
            weight(state,time,1)=wmin;
            weight(state,time,2)=prev;%This previous state is
                actualState+1
        end
    else
        weight(1,time,1)=0;
    end
end
% We have now created Trellis tree
pos=n;
message=[];
% [~,indx]=min(weight(:,n+1,1));
indx=1;
while pos>0
    from1=weight(indx,pos+1,2)-1;
    if(from1==Inf)
        from=0;
    else
        from=from1;
    end
    to=indx-1;
    msg=InputForThistransition(from,to,Diagram);
```

```
    message=[msg,message];
    indx=from1+1;
    pos=pos-1;
end
end
```

Part 3) Additional functions used in the decoders ("InputForThisTransition" and "Previous State"):

```
function inp=InputForThistransition(initialState,finalState,Diagram
    )
[noOfInpCombn,~,~]=size(Diagram);
lenOfInput=log2(noOfInpCombn);
for inpVal=1:noOfInpCombn
    if(Diagram(inpVal,initialState+1,2)==finalState)
        % inp=de2bi(inpVal-1,lenOfInput,"left-msb");
        inp=flip(bitget(inpVal-1,1:lenOfInput));
    end
end
end


function [prevState,output]=PreviousState(state,Diagram)
[noOfInpCombn,u,~]=size(Diagram);
prevState=[];
output=[];
for s=1:u
    for in=1:noOfInpCombn
        if(Diagram(in,s,2)==state)
            prevState=[prevState,s-1];
            output=[output,Diagram(in,s,1)];
        end
    end
end
end
```

6.1.7 Branch Metrics for Hard and Soft Decoders (Hamming, Euclidean and Manhattan distances)

Part 1) Hamming Distance

```
function z=hammingDistance(rcvdOutput,output,OutLen)
% rcvdOutput=de2bi(rcvdOutput,OutLen,"left-msb");
rcvdOutput=flip(bitget(rcvdOutput,1:OutLen));
% output=de2bi(output,OutLen,"left-msb");
output=flip(bitget(output,1:OutLen));
e=xor(rcvdOutput,output);
z=0;
for i=1:OutLen
    if(e(i)==1)
        z=z+1;
    end
end
```

```
end
```

Part 2) Euclidean Distance

```
function z=euclidDistance(rcvdOutput,output,OutLen,Band)
% output=de2bi(output,OutLen,"left-msb");
output=flip(bitget(output,1:OutLen));
output=(Band-1)*output;
dist=rcvdOutput-output;
% z1=abs(dist);
% z1=z1.^3;
% z=sum(z1);
% z=z^(1/3);
% z=sum(abs(dist)); % Manhattan Distance
distT=transpose(dist);
z=dist*distT; % This is used for euclidean distance
z=sqrt(z); % sqrt is a good measure
% Above is a good measure of distance
% Below are other measures of distance
% z=exp(z); % This is also a good measure
% e=exp(1);
% c=log10(e);
% z=log10(z)/c; % This is not a good metric as it numbs down the
    difference
end
```

Part 3) Manhattan Distance

```
function z=manhattanDistance(rcvdOutput,output,OutLen,Band)
output=flip(bitget(output,1:OutLen));
output=(Band-1)*output;
dist=rcvdOutput-output;
z=sum(abs(dist)); % Manhattan Distance
end
```