CT111 Project Convolutional Coding

Student Name: Kathan Sanghavi

Student ID: 201901053

Date: 15-4-2020

Honor Code

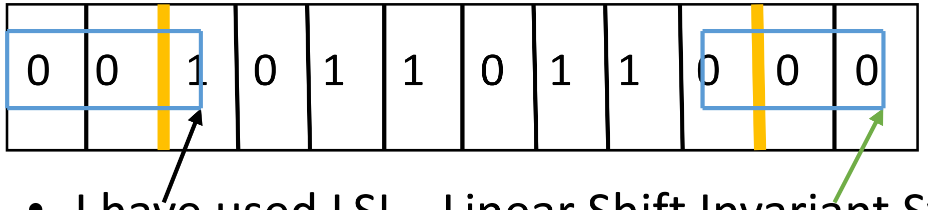
I declare that,

- The work that I am presenting is my own work.
- I have not copied the work (the code, the results, etc.) that someone else has done.
- Concepts, understanding and insights I will be describing are my own.
- I make this pledge truthfully. I know that violation of this solemn pledge can carry grave consequences.
- References: Link to References Slide(slide 19th)

Your signature

Kathan Sanghavi

Encoder Concept



* main message is between yellow lines.

- I have used LSI – Linear Shift Invariant System based approach.
- In this concept,
  - K-1 zeros are appended to message in the beginning and in the end as shown in the figure, to get terminating state to all zero.
  - As arrow shown in figure moves from original message starting position (shown in figure by black arrow) in the right direction till the last zero (shown by green arrow) K bits from that position in the left direction are taken. Ex. For K=3, for black arrow bits are taken as 100 and for green bits are taken as 000. Taken bits are multiplied with generator polynomial to generate parity bits.
- My code can work for arbitrary constraint lengths, arbitrary generator polynomials, arbitrary length of input bits and arbitrary r.
- Time complexity and space complexity are O(length of input bits). Code approximately uses space equal to size of encoded Message, which is number_of_generator_polynomials*(k+K-1) (k is length of input bits). For 10^5 input bits encoder takes 0.084 second.

Encoder Code Snippet

```
K=3;  m = [zeros(1,K-1),message,zeros(1,K-1)]; number_of_generators = 2; encodedMessage = zeros(1,number_of_generators*(k+K-1))-1;
j=0; G1=[1,1,1]; G2=[1,0,1]; G = zeros(number_of_generators,K); G(1,:) = G1; G(2,:) = G2; elements =  zeros(1,K);

for i = K:length(m)
        for ii = 1:K                                    % K = 3 for basic
                elements(ii) = m(i-ii+1);
        end
        for ii = 1:number_of_generators
                j=j+1;
                encodedMessage(j) = mod(sum(elements.*G(ii,:)),2);
        end
end     % Code can be used for any value of K and any number of generator polynomial just by adding that
        %generator polynomial in G as underlined in code.

        %encodedMessage variable is the output of encoder.
```

Decoder Concept

- In the concept,

  As encoded message has starting and ending state zero (state with all zeros), the Viterbi Decoder starts from state 0 and calculates distance(hamming or euclidean) between parity bits and received bits. Distance is added to path metric and for each state path with smaller path metric is continued. Trace back starts from last state 0 and trellis is traversed in reverse and for each state transition, bit decision is taken.

- I have implemented trellis using array called parents which stores the state of 'parent state' of each state on smaller path metric path. Parents is a array of length 2^(K-1)*(length_of_original_message+3) (For basic, K=3). Path metrics are stored in array of length same as number of states (For basic, 4 states). At each step in forward direction of trellis, values of path metrics are updated.
- Trace back is done using parents array, and bit decision is taken according to state transition.
- Code can work for arbitrary input length and arbitrary generator matrix. With minor updates in code, it can work for arbitrary constraint lengths.
- Time complexity and space complexity are O(length of input bits). It takes approximately 3*size of input bits unit of space. Generating graph of Basic part with Nsim = 20000 and k = 500 (BSC + BEC + AWGN channel) including all SNRdB values takes 10 min. Soft and hard decoder for random SNRdB and Nsim = 20000,k = 500 takes 7.88seconds. (Above results are taken by generating 500 bit message with equal probability for 0 and 1, then encoding it and passing it through all three channels, then decoding it, then finding number of bit errors. In total, passing message through channel and decoding procedure is done Nsim*(total number of all SNRdB values) times.)

Decoder:
Branch Metric Calculation Code Snippet[1]

Hard Decoder: % this code is contained inside loop which iterates through all pairs from received code.

```
temp = zeros(1,2*4)-1;   % temp contains possible values to update path metrics according to smaller value rule.
twoBits = [receivedCode(i),receivedCode(i+1)];
    if sum(twoBits)<0                  % for BEC channel erased bit is replace by -5.
      if sum(twoBits) == -10
        for k = 1:4
          temp(2*k-1) = pathMetric(k) + 2;     % here 2 is Branch Metric.
          temp(2*k) = pathMetric(k) + 2;
        end
      else
        for k = 1:4
          temp(2*k-1) = pathMetric(k) + 1;     % here 1 is Branch Metric.
          temp(2*k) = pathMetric(k) + 1;

     end
```

(slide 25th) Code continues on this slide.

Decoder:
Path Metric Calculator Code Snippet[1]

```
while i<l_r
    twoVolts = [receivedCode(i),receivedCode(i+1)];
[   for k = 1:4
      temp(2*k-1) = pathMetric(k) + sum((twoVolts-(2*pairities(2*k-1,:)-1)).^2);
      temp(2*k) = pathMetric(k) + sum((twoVolts-(2*pairities(2*k,:)-1)).^2);
   end          ]
    for x = 1:4
      if temp(x)-temp(x+4)<0
          location_min = x;
      else
          location_min = x+4;
```

% this part of code(between []) is specific to soft decoder, this part of hard decoder is in slide no. 6.

% pairities is a array which contains parity bits in particular order. This array is generated at starting of decoder according to generator matrix.

% location_min is used for saving the state from which smaller path metric has come.

(slide 26th) Code continues on this slide.

Decoder:
Trellis Traceback and Bit Decision Code Snippet[1]

```
location = 4*(l_e+3)-3;     % traceback starts from state 0, this is known as encoder terminates in state 0.
  state = 0;
  parent = parents(location);
  i = l_e+2;
  while i~=l_e
    location = parent;
    parent = parents(location);
    state = mod(location-1,4);
    i = i-1;
  end             % these bits were appended to terminate encoder at state 0. Real message starts from next state transitions.
```

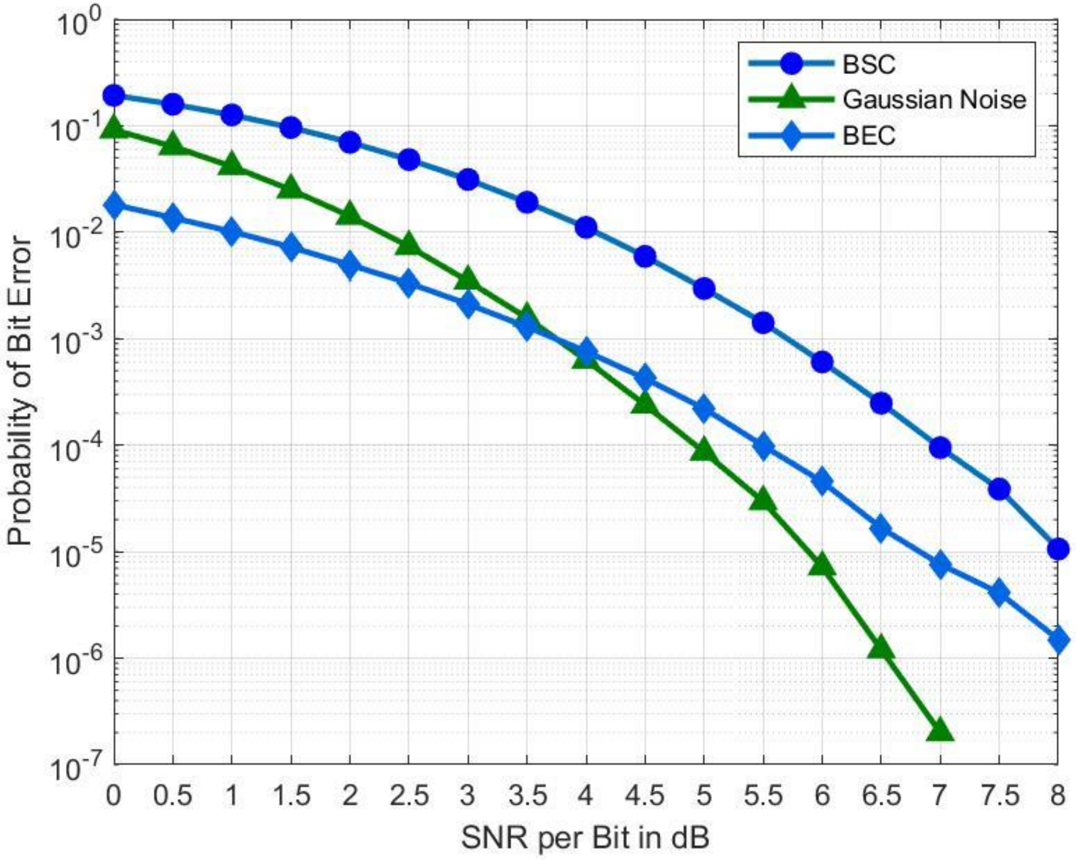
(slide 27th) Code continues on this slide.

Monte Carlo Simulator: Code Snippet[1]

```
clearvars; clc; close all; k = 500; Nsim = 20000; gammaDB = 0:0.5:8; gammaLin = 10.^(gammaDB/10); r = 1/2;
p = qfunc(sqrt(2*r*gammaLin));

pbErrBSC = zeros(1,length(gammaDB));pbErrBEC = zeros(1,length(gammaDB)); pbErrAWGN =
zeros(1,length(gammaDB));

%generating message
  p = 0.5;                              % p is probability of getting one, taking equal probabilities for 0 and 1.
  message = rand(1,k);

  for i = 1:length(message)      % length(message) = k
    if message(i) > 1-p        % (as p increase number of ones should increase.)
      message(i) = 1;
    else
      message(i) = 0;
    end
  end
```

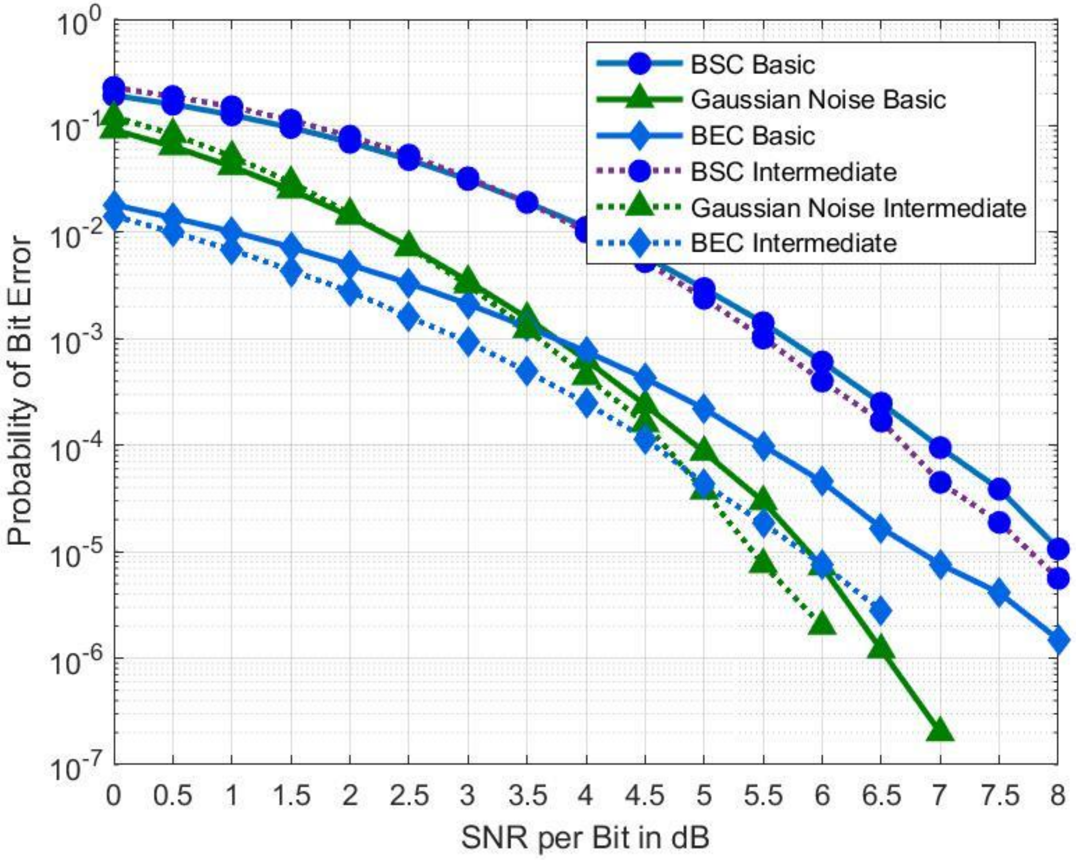
(slide 28th) Code continues on this slide

Numerical Result: Basic



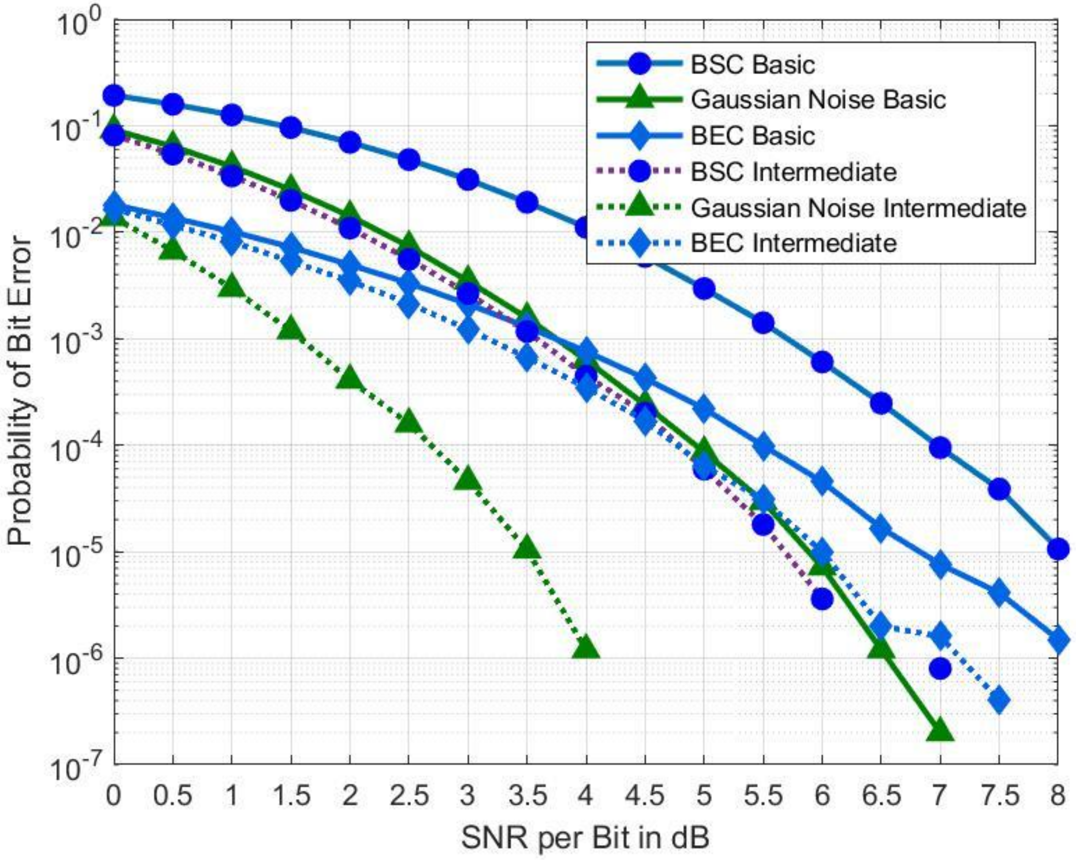
Numerical Result: Intermediate

K = 4 rate = 1/2



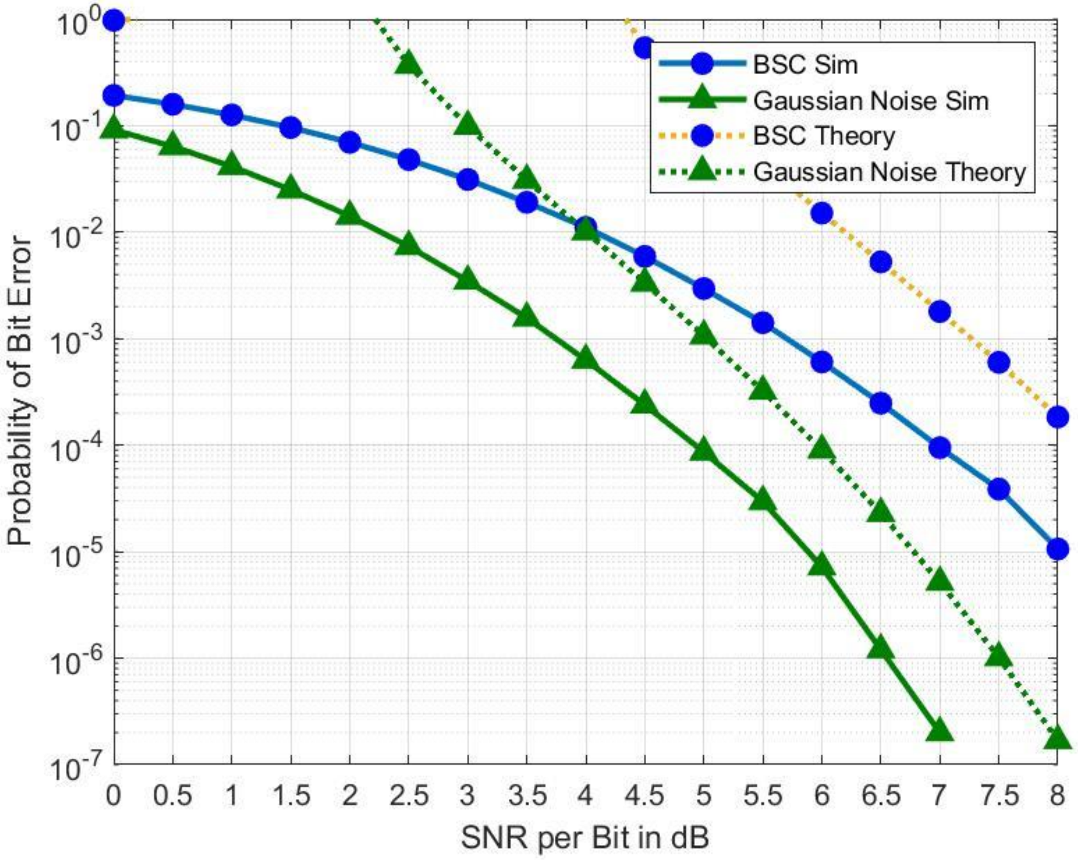
Numerical Result: Intermediate

K = 4 rate = 1/3



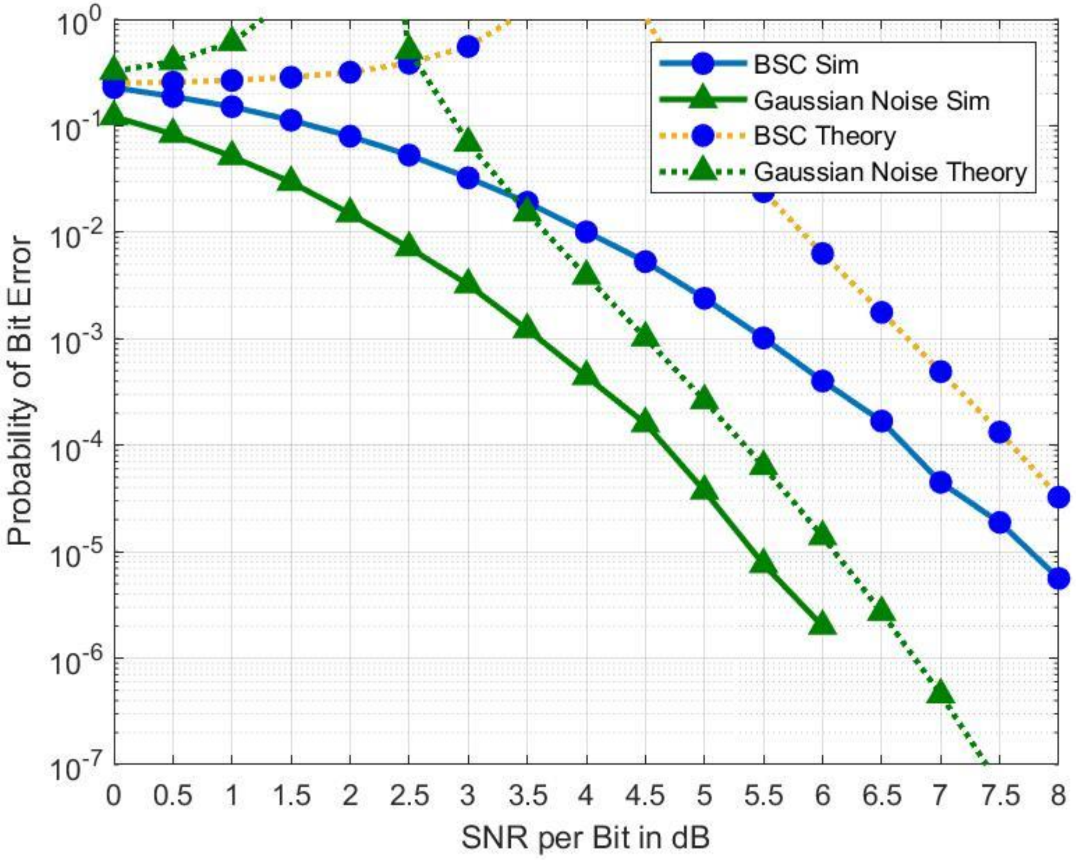
Numerical Result: Advanced Option A

K=3 r=1/2

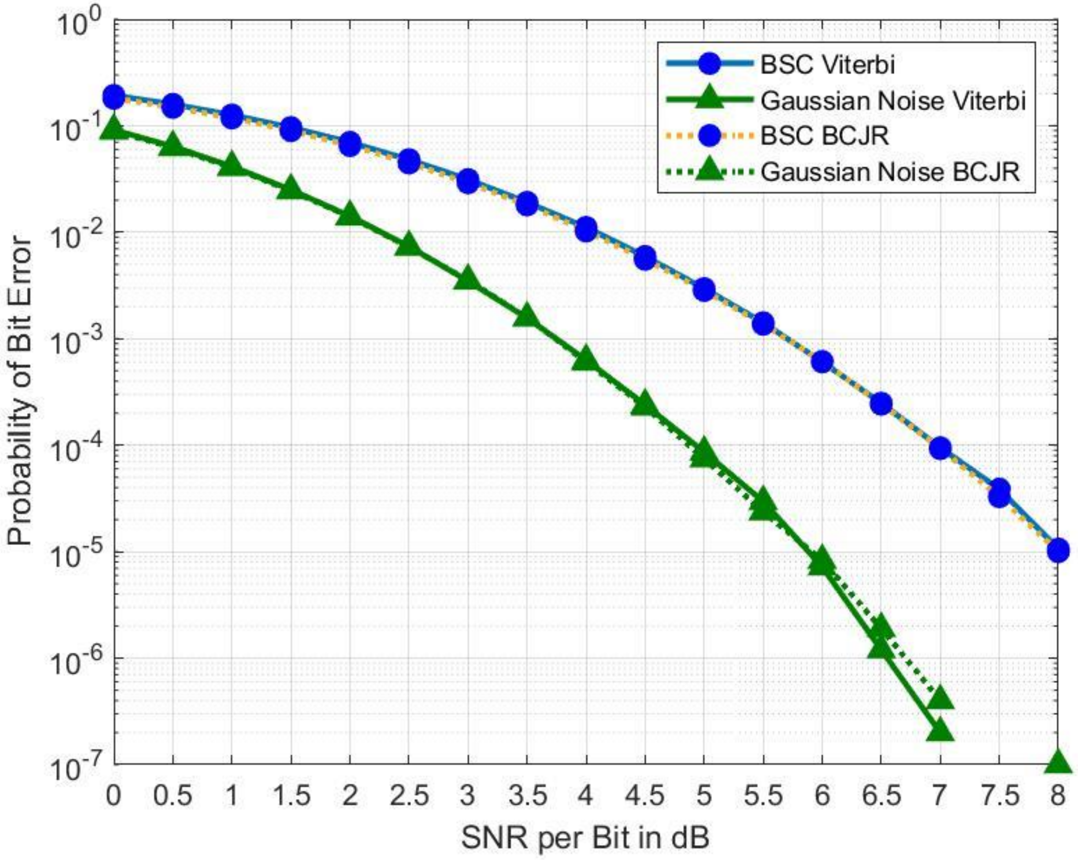


Numerical Result: Advanced Option A

K=4 r=1/2



Numerical Result: Advanced Option B[1]

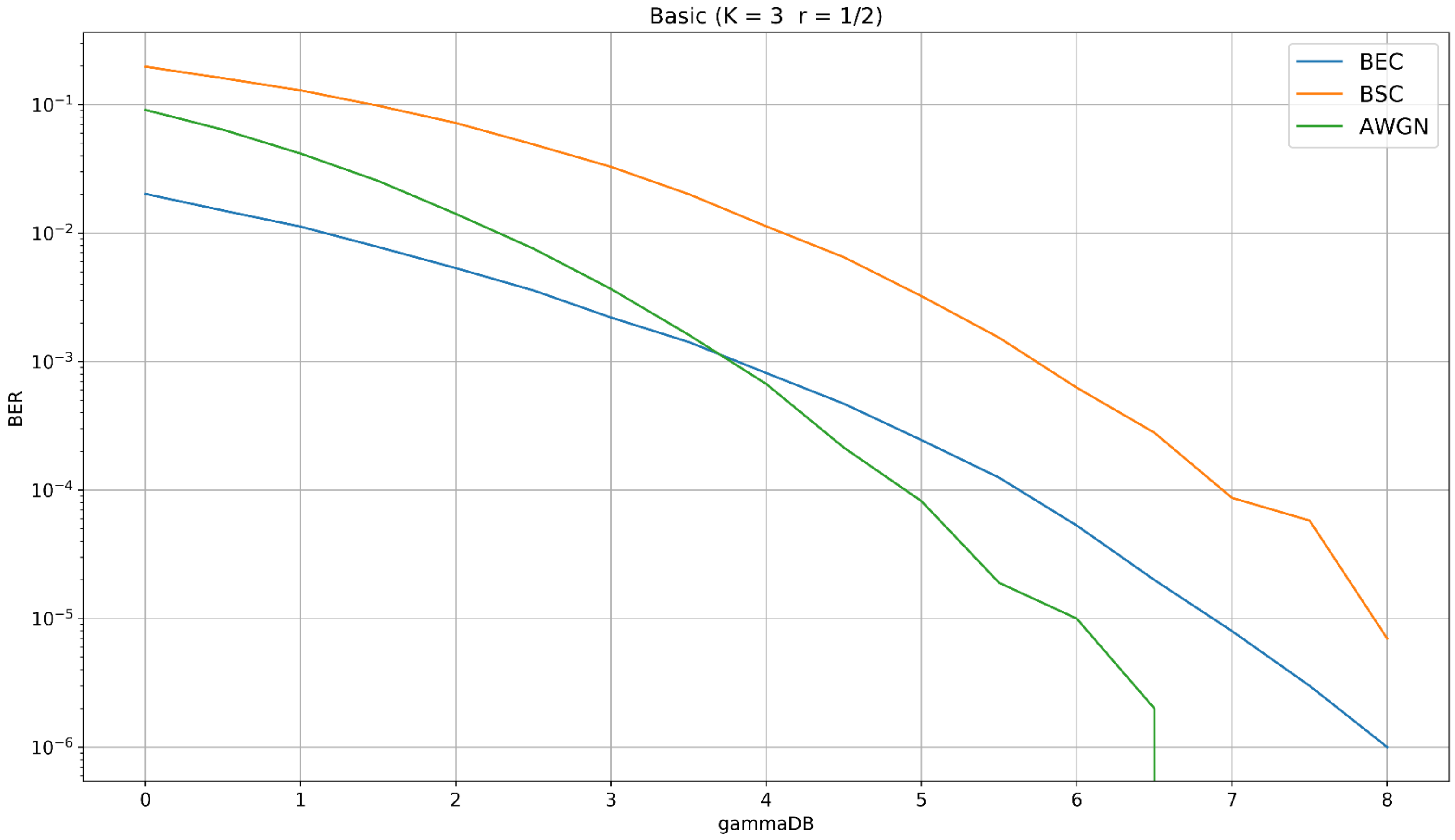


I have also done for BEC channel, I have attached that result in slide no. 24.
[Link]

Summary

- I enjoyed doing this project and learning about various decoding methods of convolutional codes.
- Algorithm can also be applied to minimum distance path search problem. A problem, in which starting point and ending point are given. We have to find a path with minimum distance to reach destination node.
- Other than Viterbi Algorithm, I would have solved decoding problem with a probabilistic approach. Algorithm: Starting with state 0 has a probability 1, calculate probability given the received bit for each state transition that can occur. For every state in next step select the path which has higher probability than the other one. Trace back can be done as we know encoder terminates on state 0.
- I have also done the basic part in python.

Basic Part Python



Above graph is generated by Python code in Spyder, with Nsim = 5000 and k = 500.

Structure for analysing Algorithms

- I have made a tree like structure by defining class node in matlab for better simulation and understanding of Viterbi Algorithm, Brute Force Algorithm or any other algorithm which uses trellis to solve path search problem.
- Structure stores information about node's parent nodes, child nodes, path metric and state at each step, which later can be used to draw simulation based on it and for analysing characteristics of different algorithms.
- This can help in debugging of algorithms and in selecting efficient algorithm for given problem.

References

- M. Valenti, \Channel Coding for IEEE 802.16e Mobile WiMAX," June 2009
- H. Balakrishnan, J. White, \MIT 6.02 DRAFT Lecture Notes, Chapter 8, Convolution Coding, Fall 2010 (Last update: October 4, 2010)
- H. Balakrishnan, J. White, \MIT 6.02 DRAFT Lecture Notes, Chapter 9, Viterbi Decoding of Convolution Codes," Fall 2010 (Last update: October 6, 2010)
- J. G. Proakis, \Digital Communications," third edition, 1994.
- M. Valenti, \Channel Coding for IEEE 802.16e Mobile WiMAX," June 2009
- L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, \Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," June 2009
- NPTEL IIT Kanpur BCJR Algorithm

Next: Appendix Slide

Link to Encoder Slide

Appendix

Derivation of the Upper Bounds on Convolutional Decoder

Derivation of the upper bound[1]

Transfers functions below are derived from solving equations based on state diagram and putting J = 1.

→ *Transfer function of K = 3, r = ½*

$T(D,N) = ND^5 / (1-2ND)$

Finding upper bound:

$(dT/dN)_{N=1} = D^5 /(1-2D)^2$

for BSC channel D = sqrt(4p(1-p))

for AWGN channel D = exp(-gammaLin*1/2) (put r=1/2)

Derivation of the upper bound[2]

→ *Transfer function of K = 4, r = ½*

$T(D,N) = (ND^6 + ND^7 - ND^8)/(1\text{-}2ND\text{-}ND^3)$

Finding upper bound:

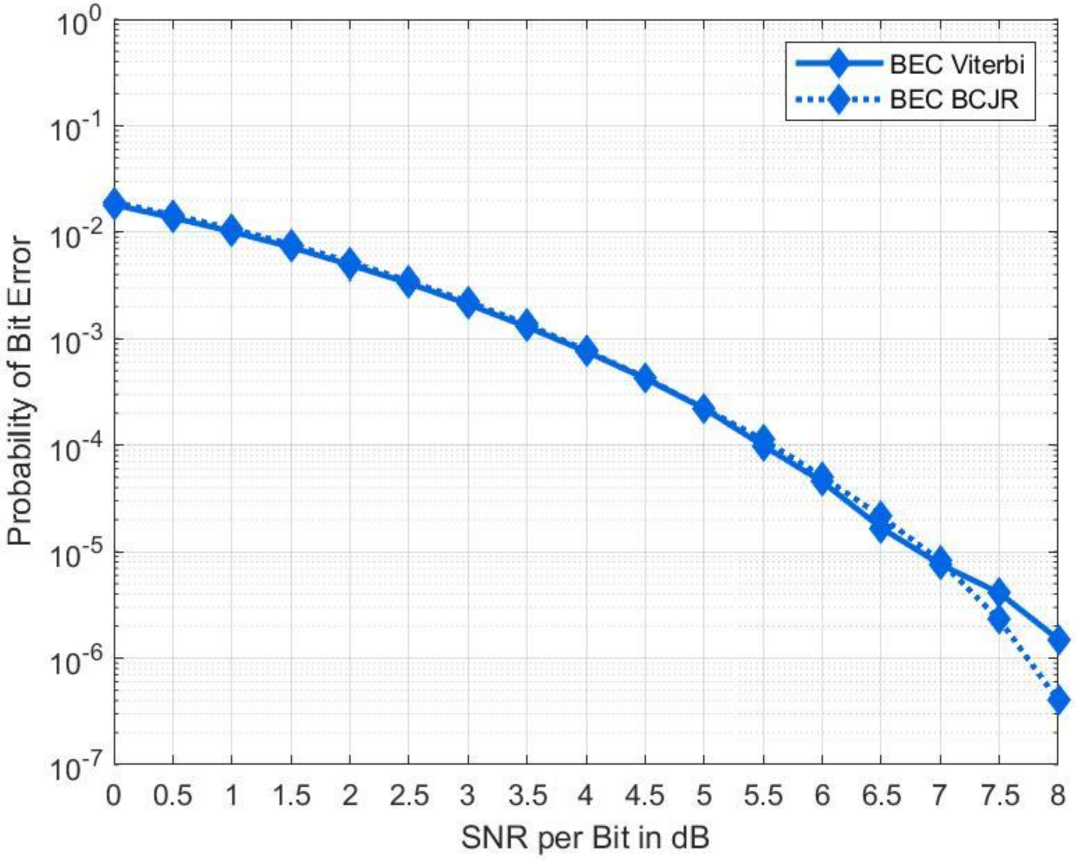
$(dT/dN)_{N=1} = (-D^8 + D^7 + D^6)/(1\text{-}D(D^2 + 2))^2$

for BSC channel D = sqrt(4p(1-p))

for AWGN channel D = exp(-gammaLin*1/2) (put r=1/2)

Linked Slides

Numerical Result: Advanced Option B²



Link to Summary

Decoder:
Branch Metric Calculation Code Snippet[2]

```
end
        else
            for k = 1:4
                temp(2*k-1) = pathMetric(k) + sum(mod(twoBits+pairities(2*k-1,:),2));
                temp(2*k) = pathMetric(k) + sum(mod(twoBits+pairities(2*k,:),2));
            end
        end
```

% underlined is branch metric, which is hamming distance between twoBits and parities. Implementing XOR using mod(A+B,2) increases simulation speed.

Soft Decoder: *% this code is contained inside loop which iterates through all pairs from received code.*

```
temp = zeros(1,2*4)-1;
twoVolts = [receivedCode(i),receivedCode(i+1)];
        for k = 1:4
            temp(2*k-1) = pathMetric(k) + sum((twoVolts-(2*pairities(2*k-1,:)-1)).^2);
            temp(2*k) = pathMetric(k) + sum((twoVolts-(2*pairities(2*k,:)-1)).^2);
        end
```

[Link to Path Metric Slide]()

Decoder: Path Metric Calculator Code Snippet[2]

```
    end
        p = 4*s+ceil(location_min/2);
        parents((4*(s+1))+x) = p;
        pathMetric(x) = temp(location_min);
      end

      i = i+2;
      s = s+1;
  end
```

% with finding smaller path metric also saving that it came from which state. For a given state there will be paths coming form two states, code will save the state from which smaller path metric has came for that state in an array parents. Parents array will be used in traceback.

% values of path metrics are updated according to smaller value rule.

% incrementing by two to get next pair from received code.

[Link to Traceback slide]

Decoder: Trellis Traceback and Bit Decision Code Snippet[2]

```
state_table = [0 0; 1 0; 0 1; 1 1];
for j = 1:l_e
      estimatedMessage(l_e-j+1) = state_table(state+1,1);
      location = parent;
      parent = parents(location);
      state = mod(location-1,4);
   end
```

% taking Bit Decision. Bit decision formula used here is derived according to table of input bit 0 or 1, state_before and state_after giving input bit.

Link to Monte-Carlo Simulation Slide

Monte Carlo Simulator: Code Snippet[2]

% generating message and then encoding can also be done in every simulation, but as bit flipping, bit erasure, adding of noise is done randomly, based on probability, one encoded message can also be used.

```
encodedMessage = encoder(message);
for i = 1:length(p)
   errBSC = 0; errBEC = 0; errAWGN = 0;
   for j = 1:Nsim
      receivedCode = channelBEC(encodedMessage,p(i));
      estimatedMessage = hardDecoder(receivedCode);
      errBEC = errBEC+ sum(xor(estimatedMessage,message));

      receivedCode = channelBSC(encodedMessage,p(i));
      estimatedMessage = hardDecoder(receivedCode);
      errBSC = errBSC + sum(xor(estimatedMessage,message));
```

Code continues on this slide

Monte Carlo Simulator: Code Snippet[3]

```
      receivedCode = channelAWGN(encodedMessage,gammaLin(i),r);
      estimatedMessage = softDecoder(receivedCode);
      errAWGN = errAWGN + sum(xor(estimatedMessage,message));
   end
   pbErrBEC(i) = errBEC/(k*Nsim);
   pbErrBSC (i) = errBSC/(k*Nsim);
   pbErrAWGN (i) = errAWGN/(k*Nsim);
end
% generating graphs (Graphs for presentation are generated from code given in template. Below code is only
for view results graphically.)
semilogy(gammaDB,pbErrBEC);hold on; semilogy(gammaDB,pbErrBSC);hold on;
semilogy(gammaDB,pbErrAWGN); legend('BEC','BSC','AWGN'); ylabel(‘Probability of Bit Error'); xlabel(‘SNRdB');
grid on; title('Basic (K = 3, r = 1/2)');
```

Link to Numerical Result: Basic Slide