CT – 111
Introduction To Communication Technology
Software Project – Sem II

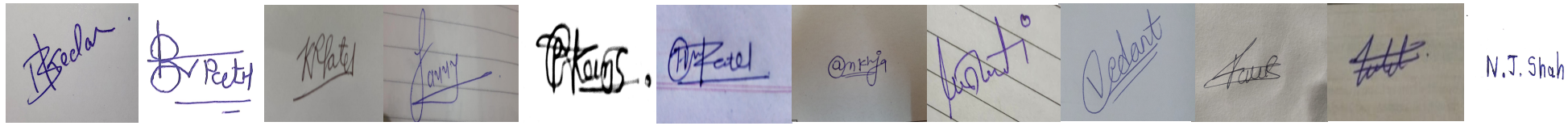
Group ID : 3

19 April 2020

Honor Code

We declare that :

- The work that we are presenting is our own work.
- We have not copied the work (the code, the results, etc.) that someone else has done.
- Concepts, understanding and insights we will be describing are our own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.
- We have taken some help from YouTube for clearing our concepts.



Contribution Table

Topics	Student Name	Student ID
Encoder - Basic	Harsh Patel	201901036
BSC Channel	Meet Prajapati	201901025
BEC Channel	Bhavya Ankhja	201901034
AWGN Channel	Vedant Parikh Purav Kansara	201901029 201901032
Decoder – Basic	Bhavya Kedar* Nilay Shah*	201901028 201901026
MonteCarlo Simulations & Result - Basic	Sanny Dhamelia Nilay Shah*	201901031 201901026
Encoders – Intermediate	Kanishk Patel Kaushal Nagani	201901035 201901033
Decoders – Intermediate	Bhuminkumar Patel Bhavya Kedar*	201901030 201901028
MonteCarlo Simulations & Result - Intermediate	Sanny Dhamelia Bhavya Kedar*	201901031 201901028
Advanced Option A	Bhavya Kedar*	201901028
Codes in C++	Paras Movaliya* Nilay Shah* Bhavya Kedar*	201901027 201901026 201901028
PPT	Bhavya Kedar*	201901028

The names that end with an asterisk '' have also made the individual project submission.

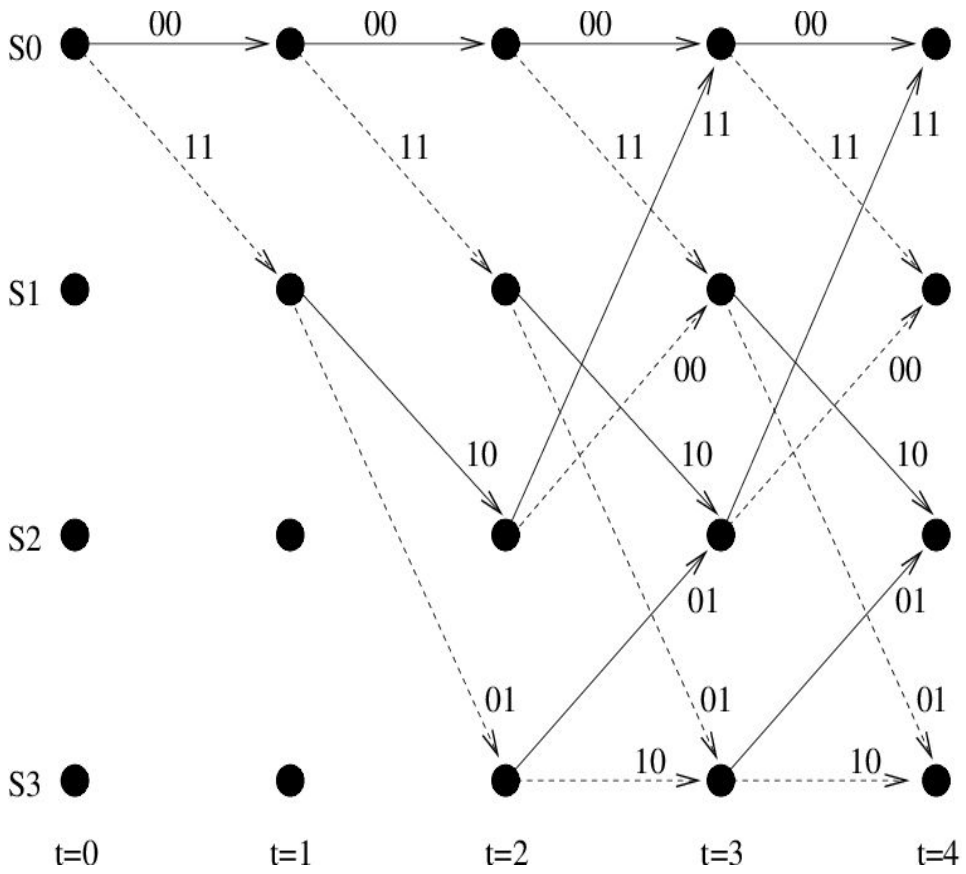
Encoder Concept

- The encoder randomly generates a binary sequence of the input length k = 500, having p(1) = p(0) = 0.5 .
- The encoder uses constraint length K = 3 and generator sequences g0 = [1 0 1] and g1 = [1 1 1]. A tail of 2 zeros is kept added to the input.
- The second and third bit of the shift register is set to 00 initially. The first bit from the left is set to the input bit. The corresponding output bits are computed using mod function and the generator sequences.
- After computing the output bits, the shift register is updated or shifted to the left and the right most bit is discarded. This process continues for entire input that is entered into the shift register from left and discarded from the right.
- The length of the output sequence is twice the length of input sequence as we get two output bits for every input bit. Hence the rate of the encoder is 0.5.
- The code runs quickly it takes 0.0031s for k = 500.

Encoder Code

```
function [input_encoder,output_encoder] = ConvolutionalEncoder(k)
   input_encoder = zeros(1,k+2);    % k = length of information bits.
   % Length of input_encoder is taken k+2 because it will require a tail
   % of two zeros to finish encoding at state = 00.
   input_encoder(1:k) = randi([0,1],1,k);
   % The above statement generates a random input sequence of 0s and 1s
   % using rand function. The initial state of the DFA is 00.
   output_encoder = zeros(1,2*(k+2));
   K = 3;
   ShiftReg = zeros(1,K);
   for i = 1:length(input_encoder)
      ShiftReg(1) = input_encoder(i);
      output_encoder(2*i-1) = mod(ShiftReg(1)+ShiftReg(2)+ShiftReg(3),2);
      output_encoder(2*i) = mod(ShiftReg(1)+ShiftReg(3),2);
      ShiftReg(3) = ShiftReg(2);
      ShiftReg(2) = ShiftReg(1);
   end
   %At the end of the above given for loop, the output_encoder is the complete
   %encoded sequence of input_encoder. input_encoder and output_encoder
   %are returned at the termination of the function.
end
```

Decoder Concept



- The decoder is based on the Viterbi Algorithm, it begins decoding from state S0 (or A) and moves a step to the right for every two bits of input.
- As the first two steps of the decoder are fixed, programming for those is separately done in my decoder code.
- As the decoder proceeds to each step taking an input of two bits, branch metrics of all the four states are calculated. The succeeding two paths of all the four states are compared.
- The paths that have lesser path metric are chosen and their path is stored, which is later useful for back tracking.
- After completing the above process for the entire input, the path metric of all the four paths are compared and the path with the lesser path metric is chosen for back tracking.
- In the back tracking the paths are examined from the tail, two bits at a time, these bits are used to back track to the correct path and note their outputs.
- My decoder is completely based on the given Trellis diagram. The computations for path matrix and branch matrix in every step is not done separately, hence I have shown my code completely in the following slides. I hope the comments are helpful.

Decoder Code

```
function [output_decoder] = ConvolutionalDecoder(inputSeq,channel)
   % The function intakes the input sequence ( inputSeq ) and the channel
   % (1-BSC , 2-BEC, 3-AWGN Channel) from which it has been passed.
   % Here consider A, B, C & D as the states 00, 01, 10 & 11 respectively
   % of the Trellis diagram.
   % Here I have declared four paths that will store the steps or input of
   % the Trellis diagram that can lead to the corresponding state at every
   % step (point of time) of decoding.
   pathA = zeros(1,length(inputSeq));
   pathB = zeros(1,length(inputSeq));
   pathC = ones(1,length(inputSeq));
   pathD = ones(1,length(inputSeq));
   % pathMetric stores the hamming distance or euclidian distance based on
   % the channel which will be used to get the surviving path at every
   % instance of decoding.
   pathMetric=[0 0 0 0];
   % As the first 4 paths are fixed in the Trellis diagram, the calculation
   % for them is done before starting the for loop.
   % pathB(1:2) is updated to [1 1] because in the first step in the
   % Trellis diagram to reach state b, the input is [1 1]. Also the
   % corresponding output is stored in outputB.
   pathB(1:2)=[1,1];
      if inputSeq(1)~=-10 && inputSeq(2)~=-10
       if channel == 3  %AWGN Channel
          branchMetric = euclidDist(inputSeq(1:2),[1 1]);
       else          %BSC or BEC Channel
          branchMetric = hammingDist(inputSeq(1:2),[1 1]);
       end
      else
       branchMetric = 10;  %For erasures found the Branch Metric is set to 10.
      end
   % pathMetric(2) is updated to pathMetric(1)+distance because in the
second
   % step, state b = 01 can be reached from state a = 00.
   % The same process as given above is carried out for other 3 paths.
   pathMetric(2) = pathMetric(1)+branchMetric;
   if inputSeq(1)~=-10 && inputSeq(2)~=-10
      if channel == 3
         branchMetric=euclidDist(inputSeq(1:2),[-1 -1]);
      else
         branchMetric=hammingDist(inputSeq(1:2),[0 0]);
      end
   else
      branchMetric = 10;
   end
   pathMetric(1) = pathMetric(1) + branchMetric;
```

Decoder Code

```
if inputSeq(3)~=-10 && inputSeq(4)~=-10
    if channel == 3
        branchMetric=euclidDist(inputSeq(3:4),[1 -1]);
    else
        branchMetric=hammingDist(inputSeq(3:4),[1 0]);
    end
  else
    branchMetric= 10;
  end
  pathMetric(3)=pathMetric(2)+branchMetric;
  pathC(3:4) = [1 0];
   if inputSeq(3)~=-10 && inputSeq(4)~=-10
    if channel == 3
        branchMetric=euclidDist(inputSeq(3:4),[-1 1]);
    else
        branchMetric=hammingDist(inputSeq(3:4),[0 1]);
    end
  else
    branchMetric = 10;
  end
  pathMetric(4) = pathMetric(2) + branchMetric;
  pathD(3:4) = [0 1];
```

```
if inputSeq(3)~=-10 && inputSeq(4)~=-10
    if channel == 3
        branchMetric=euclidDist(inputSeq(3:4),[1 1]);
    else
        branchMetric=hammingDist(inputSeq(3:4),[1 1]);
    end
  else
    branchMetric = 10;
  end
  pathMetric(2) = pathMetric(1) + branchMetric;
  pathB(3:4) = [1 1];
   if inputSeq(3)~=-10 && inputSeq(4)~=-10
    if channel == 3
        branchMetric=euclidDist(inputSeq(3:4),[-1 -1]);
    else
        branchMetric=hammingDist(inputSeq(3:4),[0 0]);
    end
  else
    branchMetric = 10;
  end
  pathMetric(1) = pathMetric(1) + branchMetric;
```

Decoder Code

```
%After finishing the calculations for the first two steps of Trellis
   %diagram, for the remaining steps a for loop is run.
   %Note : As the calculation for the first four bits is already done, the
   %loop begins with i = 5
   for i=5:2:length(inputSeq)
      if inputSeq(i)~=-10 && inputSeq(i+1)~=-10
         if channel == 3 %Here for the gausian channel, BRANCH MATRIX
                        %for all the four paths are calculated.
            Dist_00=euclidDist(inputSeq(i:i+1),[-1 -1]);
            Dist_01=euclidDist(inputSeq(i:i+1),[-1 1]);
            Dist_10=euclidDist(inputSeq(i:i+1),[1 -1]);
            Dist_11=euclidDist(inputSeq(i:i+1),[1 1]);
         else          %Here for other channels, BRANCH MATRIX
                       %for all the four paths are calculated.
            Dist_00=hammingDist(inputSeq(i:i+1),[0 0]);
            Dist_01=hammingDist(inputSeq(i:i+1),[0 1]);
            Dist_10=hammingDist(inputSeq(i:i+1),[1 0]);
            Dist_11=hammingDist(inputSeq(i:i+1),[1 1]);
         end
      else             %If erasures are found the BRANCH MATRIX for all the
                       %paths is set to 3.
         Dist_00 = 10;Dist_01 =10;Dist_10 = 10;Dist_11 = 10;
      end
      %pmA is set to the minimum of the path metrices of two possible paths
to          %state A(00).
pmA = min(Dist_00+pathMetric(1),Dist_11+pathMetric(3));
      %The pathA is updated to the path with lesser path metric using this if
statement.
      if Dist_00+pathMetric(1) <= Dist_11+pathMetric(3)
         pathA(i:i+1) = [0 0];
      else
         pathA(i:i+1) = [1 1];
      end
      %The same process is carried out for all the states.
      pmB = min(Dist_11+pathMetric(1),Dist_00+pathMetric(3));
      if Dist_11+pathMetric(1) <= Dist_00+pathMetric(3)
         pathB(i:i+1) = [1 1];
      else
         pathB(i:i+1) = [0 0];
      end
      pmC=min(Dist_10+pathMetric(2),Dist_01+pathMetric(4));
      if Dist_10+pathMetric(2) <= Dist_01+pathMetric(4)
         pathC(i:i+1) = [1 0];
      else
         pathC(i:i+1) = [0 1];
      end
      pmD=min(Dist_01+pathMetric(2),Dist_10+pathMetric(4));
      if Dist_01+pathMetric(2) <= Dist_10+pathMetric(4)
         pathD(i:i+1) = [0 1];
```

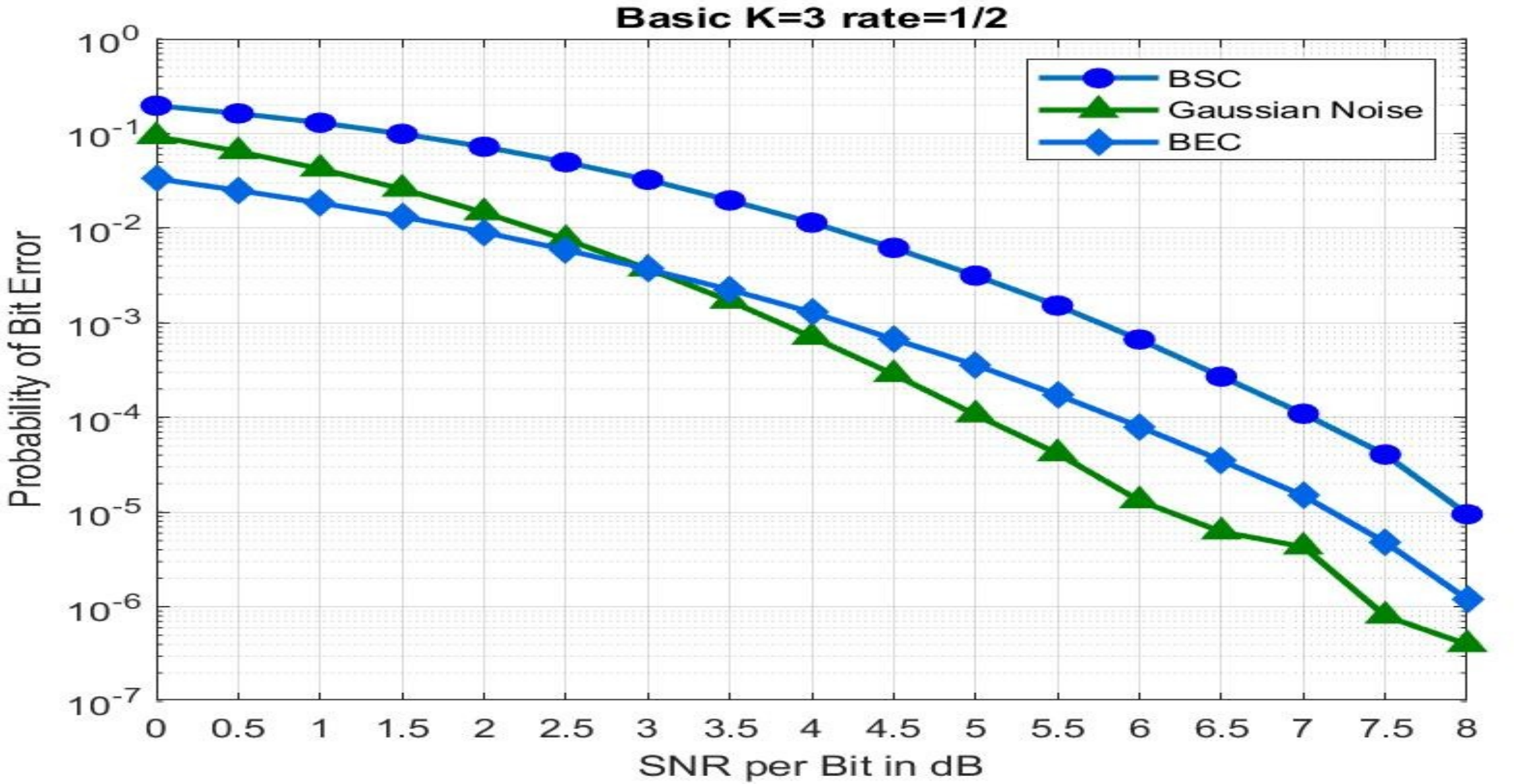
Decoder Code

```
else
        pathD(i:i+1) = [1 0];
    end
        %After selecting the 4 paths, the path metric of all the paths
        %are updated.
        pathMetric(1)=pmA; pathMetric(2)=pmB;
        pathMetric(3)=pmC; pathMetric(4)=pmD;
 end
 %After finishing the above process, the path with the minimum path
 %metric is found using below given for loop and its index is stored in
 %the variable index.
 %Hence index = 1 implies pathA, 2 implies pathB and so on.
 index=1;
 for i=2:length(pathMetric)
    if pathMetric(i)<pathMetric(index)
       index=i;
    end
 end
 output_decoder=zeros(1,length(inputSeq)/2);
 %The below given for loop does the task of back tracking the correct
 %path using the variable index.
 for i=length(inputSeq):-2:1
    if index==1
       output_decoder(i/2)=0;   %By observation output at A will always be 0.
       if pathA(i-1:i) == [1 1]
%Based on the current path, the new index is decided.
          index=3;
elseif pathA(i-1:i) == [0 0]
          index=1;
       end
    elseif index==2
       output_decoder(i/2)=1;   %By observation output at B will always be 1.
       if pathB(i-1:i) == [1 1]
          index=1;
       elseif pathB(i-1:i) == [0 0]
          index=3;
       end
    elseif index==3
       output_decoder(i/2)=0;   %By observation output at C will always be 0.
       if pathC(i-1:i) == [1 0]
          index=2;
       elseif pathC(i-1:i) == [0 1]
          index=4;
       end
    elseif index==4
       output_decoder(i/2)=1;   %By observation output at D will always be 1.
       if pathD(i-1:i) == [1 0]
          index=4;
       elseif pathD(i-1:i) == [0 1]
          index=2;
       end
    end
  end
end
```

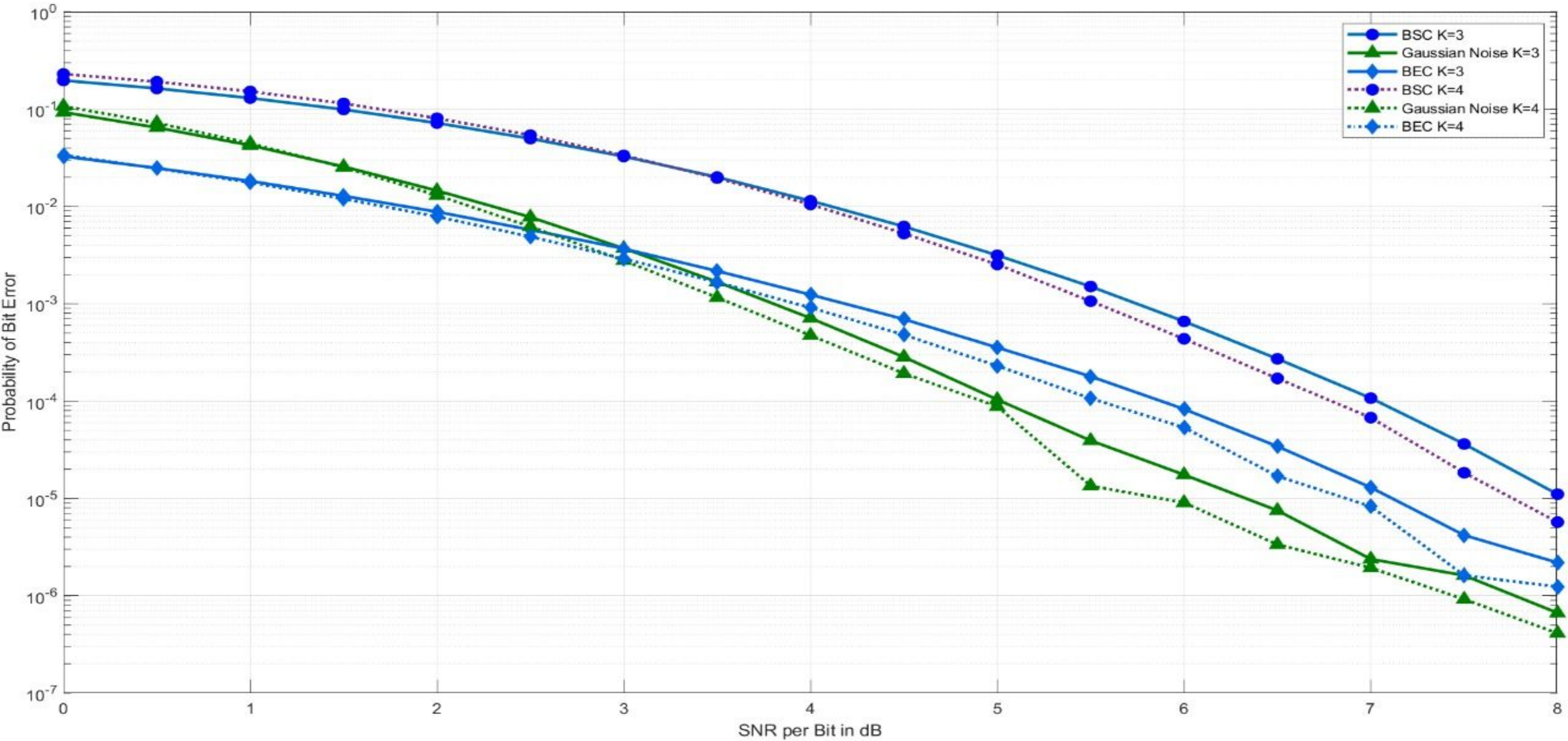
Monte Carlo Simulator Code

```
%The below given code is used for simulating 20000 simulations of input
%sequences of length 500 over all the three channels and for all SNRdb
%ranging from 0 to 8.
clearvars;
Nsim = 20000;                    %Number of simulations
k = 500;                         %Length of input sequences
rate = 0.5;                      %BitRate
SNRdb = 0 : 0.5 : 8;
SNRlin = 10.^(SNRdb/10);                 %Linear SNR
p = qfunc(sqrt(2*0.5.*SNRlin));
%Probability of error and erasure for all SNR is computed  using qFunc.
Nerror_bsc = zeros(1,length(SNRlin));
Nerror_bec = zeros(1,length(SNRlin));
%Number of errors for all the channels and for all SNR are set to zero.
Nerror_bpsk = zeros(1,length(SNRlin));
sigma2 = 1./(2*rate*SNRlin);
%sigma2 for all SNR is computed.
%The below for-loop runs for all the 20000 simulations.
for ksim = 1:Nsim
%For every simulation an input sequence of length k = 500 is generated.
   [initial_input,initial_input_encoded] = ConvolutionalEncoder(k);
   %This input sequence is used of all values of SNR.
for snr_no = 1:length(SNRlin)
      %For each SNR, the input sequence is passed through all the 3
      %channels. For BEC & BSC channel the probability corresponding to
      %current SNR is passed along with the input sequence into the
      %function. Whereas for AWGN channel the sigma2 corresponding to
      %current SNR is passed along with the input sequence into the
      %function.
      input_decoder_bec = BEC(initial_input_encoded,p(snr_no));
      input_decoder_bsc = BSC(initial_input_encoded,p(snr_no));
      input_decoder_bpsk = BPSK(initial_input_encoded,sigma2(snr_no));
      output_decoder_bec = ConvolutionalDecoder(input_decoder_bec, 2);
      output_decoder_bsc = ConvolutionalDecoder(input_decoder_bsc, 1);
      output_decoder_bpsk = ConvolutionalDecoder(input_decoder_bpsk, 3);
      %The outputs of all the channels are then compared to the initial
      %input sequence and the number of bits in error are added to the
      %Nerror corresponding to that channel and that SNR.
      Nerror_bec(snr_no)  = Nerror_bec(snr_no) +
hammingDist(initial_input,output_decoder_bec);
      Nerror_bsc(snr_no)  = Nerror_bsc(snr_no) +
hammingDist(initial_input,output_decoder_bsc);
      Nerror_bpsk(snr_no)  = Nerror_bpsk(snr_no) +
hammingDist(initial_input,output_decoder_bpsk);
   end
end
prob_BER_bec = Nerror_bec/(Nsim*k);
prob_BER_bsc = Nerror_bsc/(Nsim*k);
prob_BER_bpsk = Nerror_bpsk/(Nsim*k);
```

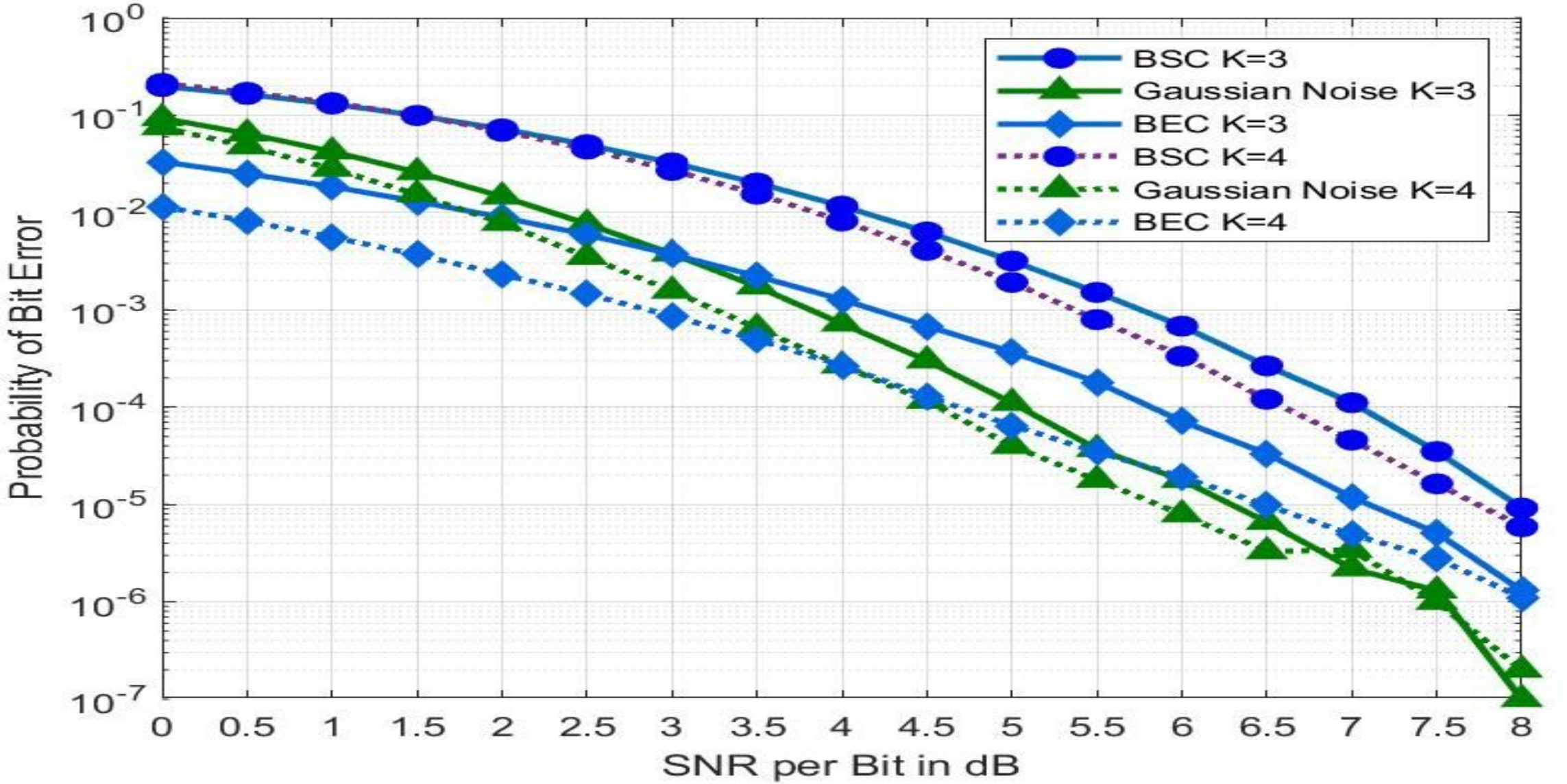
Basic : Result



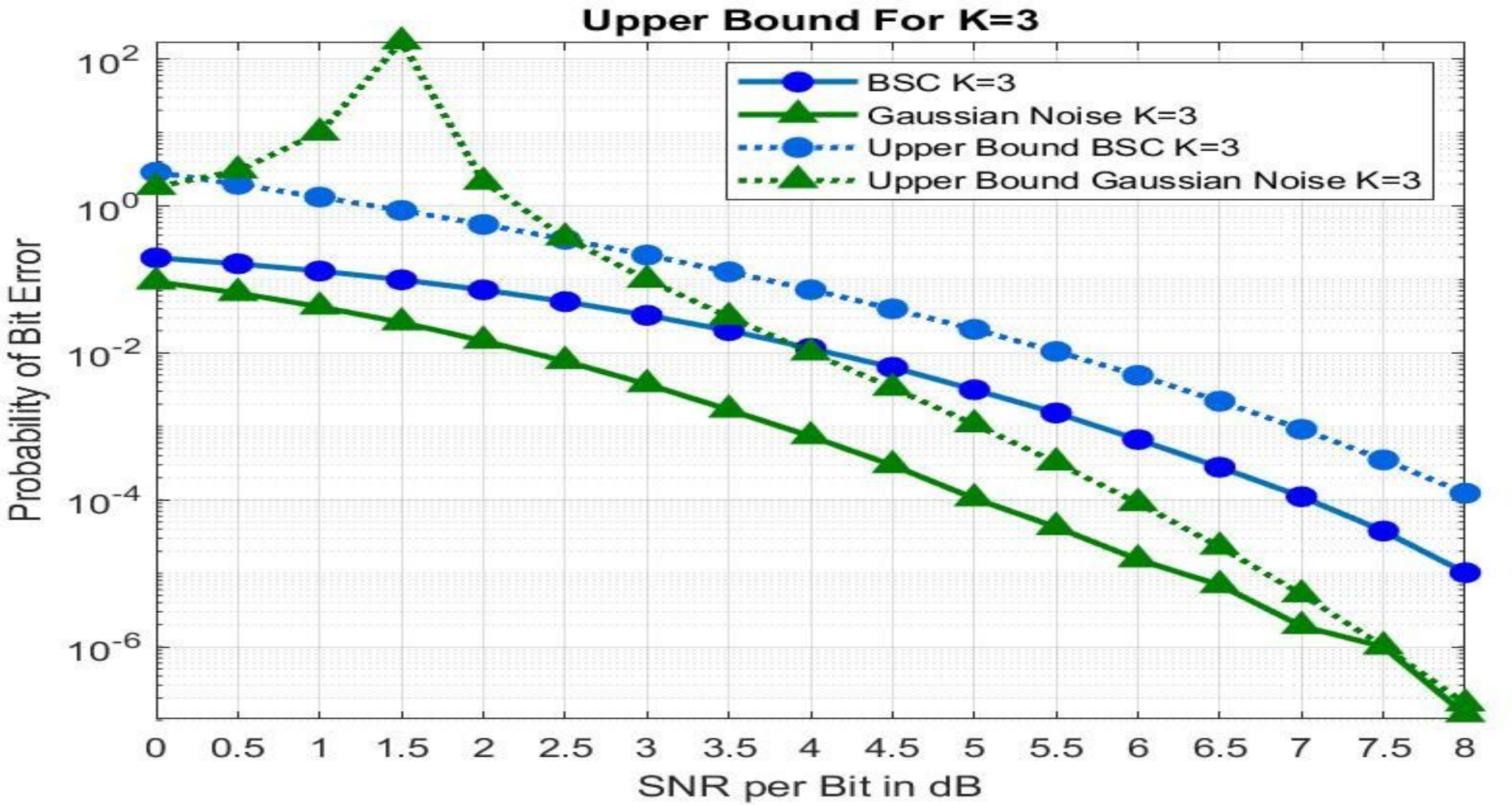
Intermediate Result



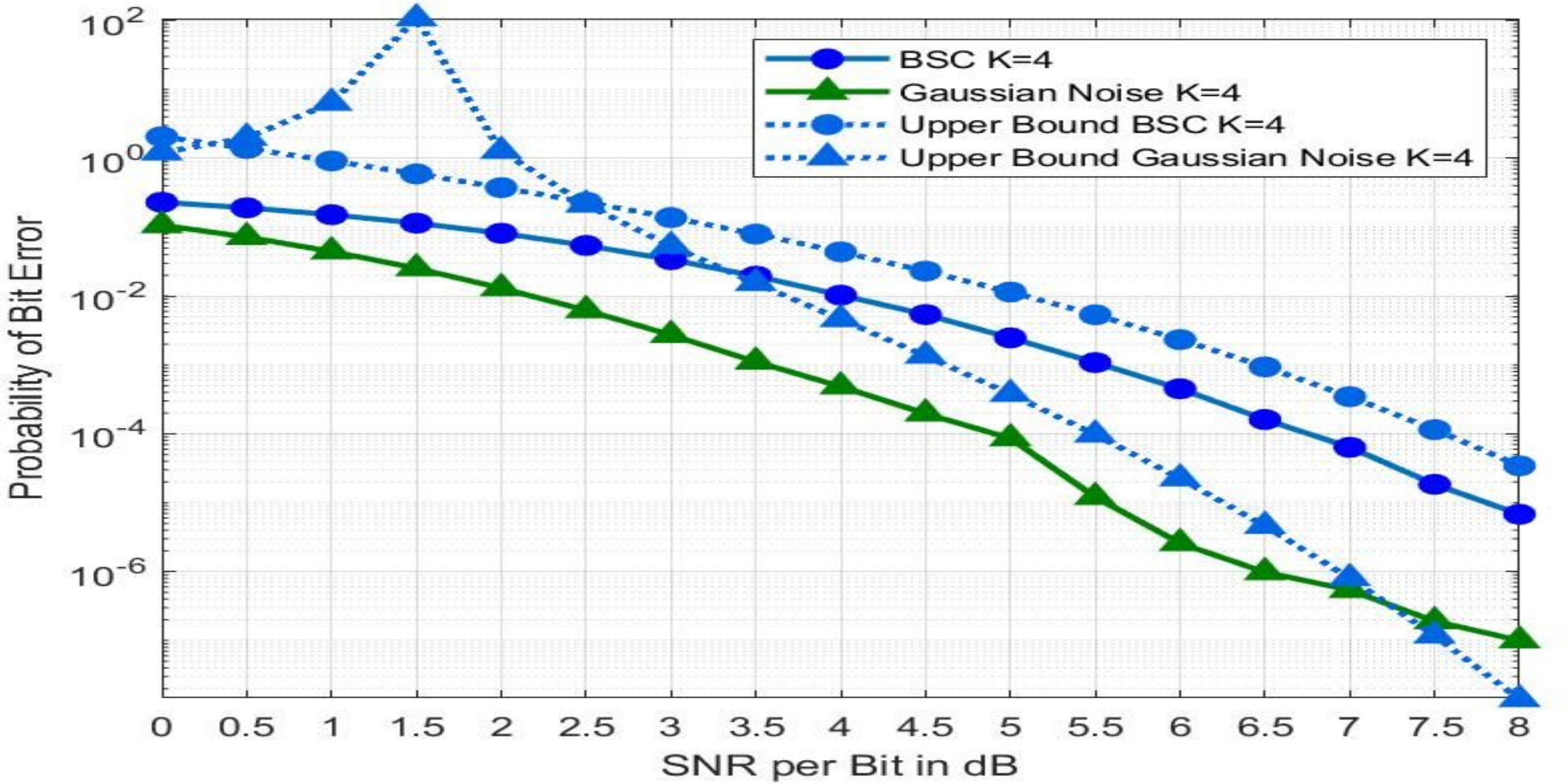
Intermediate Result



Advanced Option A Result



Advanced Option A Result



Advanced Option B

Not Attempted.

Summary

- The project was not very hard but it was very time consuming and required a very good concentration, especially for debugging the decoders. Online collaboration of the group was a big challenge.
- One of the group members, Bhavya Kedar made a different decoder in C++, that took a lot of time to compute because it generated all the 2^k possible paths using a binary tree and then generated the output by back tracking from a leaf to the root node that had least path metric. But the runtime of the code was exponential so it took around 3 secs for getting an output of length k = 18. But as it compared the input with all the possible paths the output was more accurate. This is what I would have used in substitution to the Viterbi Algorithm but taking k = 15. I have attached a link to this code in the references.
- Another group member Paras Movaliya tried to make all the basic encoder, decoder and Monte Carlo Simulator in C++. The link to the code has been provided in the references.
- Nilay Shah has made basic encoder and decoder in C++. The link to the code has been provided in the references.
- Overall group work did not turn out so well as we expected because we needed to do it online, yet we somehow managed to do so.

References

- Bhavya Kedar's C++ Decoder Code :https://drive.google.com/open?id=1E7zlGnZDXN8YQu2exBDbG__3M37xjgIg
- Paras' C++ Program :https://docs.google.com/document/d/1-t3NVrCdlaW1NMP6gdsIhv8Fx9XkWPNS8ICbAHSfK-k/edit?usp=sharing
- Nilay's C++ Program :https://docs.google.com/document/d/1iIaRu-wtCvXCMyy7oa97TzgHM7gWyEycaLwFN9YzoVQ/edit?usp=sharing
- Nilay's PPT of different approach of decoding scheme : https://drive.google.com/file/d/1NoXfxPxmJv3Z4oXo_cCwFMbMBM55tYx-/view?usp=drivesdk
- Other functions required to run our code :

  BEC : https://drive.google.com/open?id=1GN3_ERY5MtL1wZoltGg2H2NhwXG6ioZa

  BPSK : https://drive.google.com/open?id=13aYkdN9XAyjFxU6oQuAGlW8ytv8NAQwB

  BSC : https://drive.google.com/open?id=1Txq0vdSUq9oGRPO73m7__yI1ATMoqP4U

  hammingDist : https://drive.google.com/open?id=1eC41s6HK_uwxyoiyE5TTnLPxp8Stzyk3

  euclidDist : https://drive.google.com/open?id=1ay2lVmTQc0OCvpAblaleC__ta4AGl898
- YouTube Reference Links :https://youtu.be/r0hJxzJyIw8

  https://youtu.be/cK1IchFQDfU

  https://youtu.be/DM7sYJowN-A
- Image in this PPT is taken from :

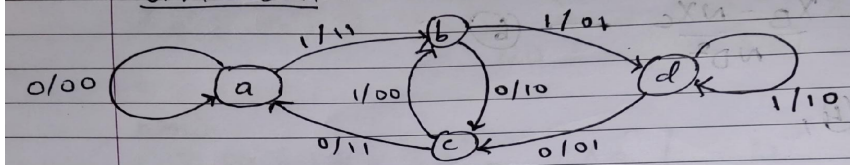
https://www.researchgate.net/profile/Juha_Plosila/publication/31595950/figure/fig4/AS:654072533221376@1532954452005/Trellis-diagram-for-rate-1-2-K3-convolutional-code.png

Appendix : Derivation of Upper Bounds

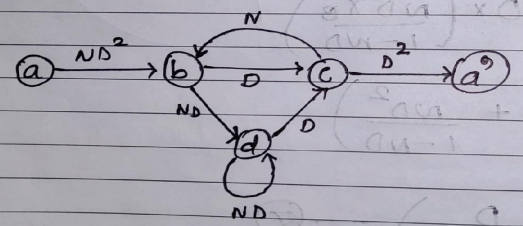
Advanced A : Proof

* $K = 3$

Normal DFA :



Turning it into D, N format where.
Power of D = Weight of the output bits
Power of N = Input bit.
Also breaking the initial state a into two diffrent states a and a' where a is initial state and a' is terminating state.



Using the above obtained DFA, the equations that can be obtained are,

$$X_A' = D^2 X_C \quad \text{---(1)}$$

$$X_B = ND^2 X_A + NX_C \quad \text{---(2)}$$

$$X_C = DX_B + DX_D \quad \text{---(3)}$$

$$X_D = NDX_B + NDX_D \quad \text{---(4)}$$

Using (2),

$$X_A = \frac{X_B - NX_C}{ND^2} \quad \text{---(6)}$$

Using (4),

$$X_D(1 - ND) = NDX_B$$

$$\Rightarrow X_D = \frac{NDX_B}{1 - ND} \quad \text{---(5)}$$

Using (5) in (3),

$$X_C = DX_B + D \times \left(\frac{NDX_B}{1 - ND}\right)$$

$$= X_B\left(D + \frac{ND^2}{1 - ND}\right)$$

$$= X_B\left(\frac{D}{1 - ND}\right) \quad \text{---(7)}$$

Using (7) in (6),

$$X_A = \frac{X_B - N\left(\frac{X_B D}{1 - ND}\right)}{ND^2} = \frac{X_B - NDX_B - NDX_B}{ND^2(1 - ND)}$$

$$\Rightarrow \frac{X_B - 2NDX_B}{ND^2(1 - ND)} \quad \text{---(8)}$$

Using (7) in (1),

$$X_A' = D^2 \times X_B \left(\frac{D}{1-ND}\right) = \frac{D^3}{(1-ND)} X_B$$

Now

$$T(D,N) = \frac{X_A'}{X_A} = \frac{X_B\left(\frac{D^3}{1-ND}\right)}{X_B\left(\frac{1-2ND}{ND^2(1-ND)}\right)}$$

$$= \frac{ND^5}{1-2ND}$$

$$\Rightarrow \frac{dT(D,N)}{dN} = \frac{D^5(1-2ND) - (-2D)(ND^5)}{(1-2ND)^2}$$

$$= \frac{D^5 - 2ND^6 + 2ND^6}{(1-2ND)^2}$$

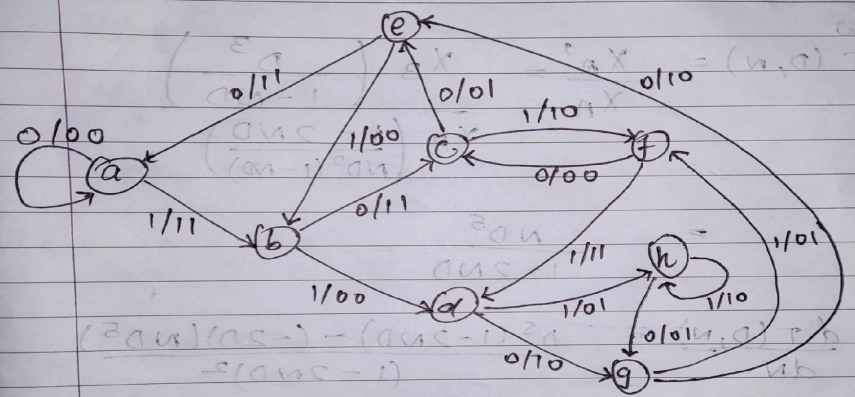
$$= \frac{D^5}{(1-2ND)^2}$$

BSC $\Rightarrow T'(D,N)_{\substack{D=\sqrt{4p(1-p)} \\ N=1}} = \frac{\left(\sqrt{4p(1-p)}\right)^5}{\left(1-2\sqrt{4p(1-p)}\right)^2}$

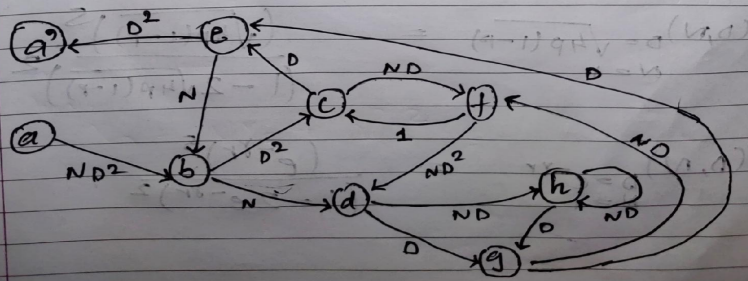
WGN $\Rightarrow T'(D,N)_{\substack{D=e^{-\gamma r} \\ N=1}} = \frac{(e^{-\gamma r})^5}{(1-2e^{-\gamma r})^2}$

* $K = 4$

Normal DFA:



Turning it into D,N format where
Powers of D = Weight of output bits
Power of N = Input bit.
Also breaking the state a into two a and a' where a is initial state and a' is terminating state.



$$X_B = ND^2X_A + NX_E$$

$$X_C = X_F + D^2X_B$$

$$X_D = NX_B + ND^2X_F$$

$$X_E = DX_G + DX_C$$

$$X_F = NDX_C + NDX_G$$

$$X_G = DX_D + DX_H$$

$$X_H = NDX_H + NDX_D$$

$$X_A' = D^2X_E$$

→ From the DFA we get above 8 equations, Solving them, we get,

$$T(D,N) = \frac{X_A'}{X_A} = \frac{ND^7 - N^2D^8 + N^2D^6}{1 - 2ND - ND^3}$$

Neglecting ND^3 we get,

$$T(D,N) \approx \frac{N^2D^6 - N^2D^8 + ND^7}{1 - 2ND}$$

$$\Rightarrow \frac{dT(D,N)}{dN} \approx \frac{(2ND^6 - 2ND^8 + D^7)(1 - 2ND) - (1 - 2ND)(2N^2D^6 - N^2D^8 + ND^7)}{(1 - 2ND)^2}$$

$$\approx \frac{2ND^6 - 2ND^8 + D^7 - 4N^2D^7 + 4N^2D^9 - 2ND^8 + 2N^2D^8 - 2N^2D^9 + 2ND^8 + 2N^2D^7}{(1 - 2ND)^2}$$

$$\approx \frac{2ND^6 - 2ND^8 + D^7 - 2N^2D^7 + 2N^2D^9 + 2N^2D^8}{(1 - 2ND)^2}$$

Hence,
for BSC, $N = 1$, $D = \sqrt{4p(1-p)}$

$$\frac{dT(D,N)}{dN} = \frac{2D^6 - 2D^8 + D^7 - 2D^7 + 2D^9 + 2D^9}{(1 - 2D)^2}$$

$$= \frac{2D^9 - 2D^8 + 2D^6 - D^7}{(1 - 2D)^2}$$

$$= \frac{2(\sqrt{4p(1-p)})^9 - 2(4p(1-p))^4 + 2(4p(1-p))^3 - (\sqrt{4p(1-p)})^7}{(1 - 2\sqrt{4p(1-p)})^2}$$