Introduction to Communication Systems

Convolutional Codes

Group 19

1 Transfer function

The distance properties and the error rate performance of a convolution code can be obtained from its state diagram. Assuming the all-zero code sequence is transmitted, the state diagram can be used to analyze the distance properties of the convolution code and evaluate its performance in terms of error rate. which is shown in figure 1.

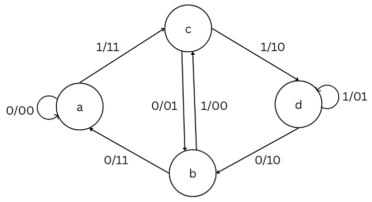


Figure 1: State diagram for kc = 3, r = (1/2)

Each branch of the state diagram is labeled with an exponent of D, where the exponent denotes the Hamming distance between the output bits on that branch and the all-zero output sequence (i.e., the transmitted bits). Additionally, state a is split into two nodes: one representing the input and the other representing the output, to facilitate the analysis. The resulting modified diagram is illustrated below in figure 2.

From this representation, we derive four state equations.

$$
\begin{aligned}
X_a &= D^0 X_a = X_a \\
X_b &= DX_c + DX_d \\
X_c &= D^0 X_b + D^2 X_a = X_b + D^2 X_a \\
X_d &= DX_d + DX_c \\
X_e &= D^2 X_b
\end{aligned}
$$

$$
\begin{aligned}
X_d - DX_d &= DX_c \\
\implies X_d &= \frac{D}{1-D} X_c \\
\implies X_b &= DX_c + DX_d \\
&= DX_c + D\left(\frac{D}{1-D}\right) X_c \\
&= X_c \left(D + \frac{D^2}{1-D}\right) \\
&= X_c \left(\frac{D(1-D) + D^2}{1-D}\right) \\
&= X_c \left(\frac{D - D^2 + D^2}{1-D}\right) \\
&= \frac{DX_c}{1-D} \\
\implies X_b &= \frac{D}{1-D}\left(X_b + D^2 X_a\right)
\end{aligned}
$$

$$
\begin{aligned}
\implies (1-D)X_b &= DX_b + D^3 X_a \\
\implies (1-2D)X_b &= D^3 X_a \\
\implies X_b &= \frac{D^3 X_a}{1-2D} \\
\implies X_e &= D^2 X_b \\
&= \frac{D^5 X_a}{1-2D}
\end{aligned}
$$

$$
\begin{aligned}
\implies T(D) &= \frac{X_e}{X_a} \\
&= \frac{D^5}{1-2D} \\
&= D^5 + 2D^6 + 4D^7 + \cdots \\
\implies T(D) &= \sum_{d=5}^{\infty} a_d D^d
\end{aligned}
$$

Here, a_d represents the number of distinct paths in the state diagram that result in a Hamming distance of d. For example, there is one path with $d = 5$, two paths with $d = 6$, four paths with $d = 7$, and so on. The smallest non-zero Hamming distance among all such paths is referred to as the minimum free distance, denoted by d_{free}. In this example, we have $d_{\text{free}} = 5$.

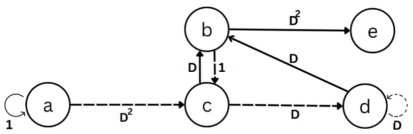


Figure 2: Revised state diagram for $k_c = 3$, $r = \frac{1}{2}$

Now we introduce a factor N into all branch transitions caused by the input bit 1 and a factor of J into into each branch of state diagram that indicates the number of branches in any given path from node a to e.
So now the state equations becomes

$$\begin{aligned} X_a &= JX_a \\ X_b &= JNDX_c + JNDX_d \\ X_c &= JX_b + JD^2X_a \\ X_d &= JNDX_d + JNDX_c \\ X_e &= JD^2X_b \end{aligned}$$

From this we obtain the transfer function

$$\begin{aligned} T(D, N, J) &= \frac{D^5J^3N}{1 - DJN(1 + J)} \\ &= D^5J^3N + D^6J^4N^2(1 + J) + ... \end{aligned}$$

This indicates that for $d = 5$ we got one path that has length of 3 and has one input bit 1 in it. That is the exponent of J indicates the length of the path that merges with all-zero path, the exponent of N indicates the number of 1's in that path and the exponent of D denotes the distance of encoded bits for that path from the all-zero sequence.

2 Probability of error in soft decoding

In decoding a block code for a memoryless channel, we computed the Euclidean distance for soft decision decoding between the received codeword and all other 2^k possible codewords. Then we select the codeword that is closest to the codeword received. For simplicity we assume that all zero codeword is transmitted.

If the coded sequence is transmitted through BPSK, then the received codeword becomes

$$r_{jm} = \sqrt{E_s}(2c_{jm} - 1) + n_{jm} \tag{1}$$

Here, r_{jm} represents the m^{th} bit of the j^{th} branch of the received codeword, c_{jm} represents the m^{th} bit of the j^{th} branch of the transmitted codeword, and n_{jm} represents noise. And E_s is the energy of the transmitted symbol.

A metric is defined for the j^{th} branch of the i^{th} path through the trellis as the logarithm of the joint probability of the sequence r_{jm} conditioned on the transmitted sequence c_{jm} for the path i

$$\mu_j^{(i)} = \log P(Y_j \mid C_j^{(i)}), \qquad j = 1, 2, 3, \ldots \tag{2}$$

Furthermore, a metric for the i^{th} path consisting of B branches through the trellis is defined as

$$PM^{(i)} = \sum_{j=1}^{B} \mu_j^{(i)} \tag{3}$$

We select the path having larger metric that is, we select the path that minimizes the probability of error. In our case channel introduce white gaussian noise to the signal, so that pdf of received sequence can be written as,

$$p\left(r_{jm} \mid c_{jm}^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{\left[r_{jm} - \sqrt{E_s}\left(2c_{jm}^{(i)} - 1\right)\right]^2}{2\sigma^2} \right\} \tag{4}$$

where $\sigma^2 = \frac{1}{2}N_0$ is the variance of the additive gaussian noise.

$$\mu_j^{(i)} = \log\left[\prod_{m=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{\left[r_{jm} - \sqrt{E_s}\left(2c_{jm}^{(i)} - 1\right)\right]^2}{2\sigma^2} \right\} \right]$$

$$\mu_j^{(i)} = \sum_{m=1}^{n} \log\left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{\left[r_{jm} - \sqrt{E_s}\left(2c_{jm}^{(i)} - 1\right)\right]^2}{2\sigma^2} \right\} \right]$$

$$\mu_j^{(i)} = \sum_{m=1}^{n} \left[\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{\left[r_{jm} - \sqrt{E_s}\left(2c_{jm}^{(i)} - 1\right)\right]^2}{2\sigma^2} \right]$$

Here, the variance and the other constants can also be ignored, so now we will just have,

$$\mu_j^{(i)} = \sum_{m=1}^{n} - \left[r_{jm}^2 - 2r_{jm}\sqrt{E_s}\left(2c_{jm}^{(i)} - 1\right) + E_s\left(2c_{jm}^{(i)} - 1\right)^2 \right]$$

Now, the term r_{jm}^2 will be same in $\mu_j^{(i)}$ for a given j^{th} branch and m^{th} bit corresponding to all possible path i . We can also conclude that $E_s\left(2c_{jm}^{(i)} - 1\right)^2 = E_s$ (irrespective of $c_{jm}^{(i)}$ is 0 or 1). In the remaining term $2\sqrt{E_s}$ is also constant. So, we will just ignore it.

The branch metric for the j^{th} branch of the i^{th} path can be expressed as

$$\mu_j^{(i)} = \sum_{m=1}^{n} r_{jm}(2c_{jm}^{(i)} - 1) \tag{5}$$

So now the path metric of the i^{th} containing B branches becomes

$$CM^{(i)} = \sum_{j=1}^{B} \mu_j^{(i)} = \sum_{j=1}^{B} \sum_{m=1}^{n} r_{jm} \left(2c_{jm}^{(i)} - 1\right) \tag{6}$$

Now, for every path merging in a state, we select that path with the largest metrics and discard all other paths. Let's assume that there is one all zero path that is the correct one denoted by i=0 and for i=1 we get an incorrect path. Now since we have transmitted all zero codeword, let's say there are d ones in received sequence means that path differs by d bits from all zero path, then let $P_2(d)$ define Probability of error in selecting the path with d ones. In our example wrong path is selected when $CM^{(1)}$ is greater than $CM^{(0)}$. So from equation (6) we can write that

$$P_2(d) = P\left(CM^{(1)} \geq CM^{(0)}\right) = P\left(CM^{(1)} - CM^{(0)} \geq 0\right) \tag{7}$$

$$P_2(d) = P\left[2\sum_{j=1}^{B} \sum_{m=1}^{n} r_{jm} \left(c_{jm}^{(1)} - c_{jm}^{(0)}\right) \geq 0\right] \tag{8}$$

Here, $(c_{jm}^{(i)})$ denotes the m^{th} coded bit of the j^{th} branch of the i^{th} path. We know that $(c_{jm}^{(1)})$ and $(c_{jm}^{(0)})$ are the same except in the d positions. So, Equation (8) can be rewritten as

$$P_2(d) = P\left(\sum_{l=1}^{d} r_l' \geq 0\right) \tag{9}$$

Where index l runs over the set of d bits in which the two paths differ and the set $\{r_l'\}$ represents the set of d received bits.

From Equation (1), along with the assumption that we have transmitted the all zero codeword, we can conclude that $\{r_l'\}$ are independent and identically distributed gaussian random variables with mean $-\sqrt{E_s}$ and variance $\frac{1}{2}N_0$. p.d.f. of $\{r_l'\}$ is defined as

$$p(r_l') = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \cdot \frac{(r + \sqrt{E_s})^2}{\sigma^2}\right) \tag{10}$$

$\mathrm{r}_l' \sim N(-\sqrt{E_s}, \sigma^2)$ Implies,

$\sum_{l=1}^{d} r_l' \sim N(-d\sqrt{E_s}, d\sigma^2)$

Now the probability of error for a $\{r_l'\}$ can be written as,

$$P(\sum_{l=1}^{d} r_l' \geq 0) = \int_0^{\infty} \frac{1}{\sqrt{2\pi d}\sigma} \exp\left(-\frac{1}{2} \cdot \frac{(r + d\sqrt{E_s})^2}{d\sigma^2}\right) dr \tag{11}$$

$$u = \frac{r + d\sqrt{E_s}}{\sqrt{d}\sigma}$$

$$du = \frac{dr}{\sqrt{d}\sigma}$$

So limits are changed to $\frac{\sqrt{dE_s}}{\sigma}$ and ∞

$$P(\sum_{l=1}^{d} r_l' \geq 0) = \int_{\frac{\sqrt{dE_s}}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-u^2}{2}\right) dr \tag{12}$$

That can be written as

$$P(\sum_{l=1}^{d} r_l' \geq 0) = Q(\frac{\sqrt{dE_s}}{\sigma}) \tag{13}$$

Now $\sigma^2 = \frac{1}{2}N_0$ and each $\{r_l'\}$ are identical and independent so,

$$P(\sum_{l=1}^{d} r_l' \geq 0) = Q(\sqrt{\frac{2E_s d}{N_0}}) \tag{14}$$

$$\boxed{P_2(d) = Q\left(\sqrt{2\gamma_b R_c d}\right)}$$

Where $\gamma_b = \frac{E_b}{N_0}$ is the received SNR per bit and R_c is the code rate.

Here, we have derived the first-probability of an event error of a path distance d from the all-zero path, but there are many possible paths with different distances that merge with the all-zero path at a given node B.

Now, we will consider how errors can occur in the decoding process. The decision is taken when two paths join together. If two paths join together and the path with the higher matric is actually the incorrect path, then the an incorrect decision is made at that point. We call such an error a node error and denote the probability of a node error as P_e. The error occurs at the place where the path first diverge.

Suppose,in the figure given below, the path diverging from the all zero path at a has a higher metric when the path merge at a'. Due to this error event happens at a. Suppose that there are error events also at b and d. Now consider the path diverging at c: even if the metric is higher (better) at c', the diverging path from c may not ultimately be selected if its metric is worse than the path emerging at b. Similarly the path emerging at d may not necessarily be selected, since the path merging at e may take precedence. This overlapping of decision paths makes the exact analysis of the bit error rate difficult.

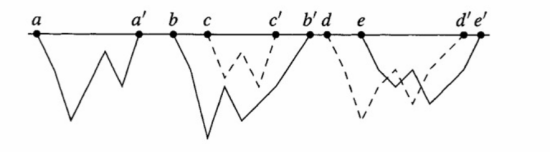


Figure 3: Figure of the error due to merging paths.

Thus we obtain an upper bound on the first-event error probability in the form

$$P_e \leq \sum_{d=d_{\text{free}}}^{\infty} a_d P_2(d) \tag{15}$$

$$P_e \leq \sum_{d=d_{\text{free}}}^{\infty} a_d Q\left(\sqrt{2\gamma_b R_c d}\right) \tag{16}$$

where a_d denotes the number of paths of distance d from the all-zero path that merge with the all-zero path for the first time.

Now, the upper-bound of the Q function is given as:

$$Q\left(\sqrt{2\gamma_b R_c d}\right) \leq e^{-\gamma_b R_c d} = D^d\Big|_{D=e^{-\gamma_b R_c}} \tag{17}$$

If we use Equation (17) in Equation (16),

$$P_e \leq \sum_{d=d_{\text{free}}}^{\infty} a_d D^d \tag{18}$$

$$P_e < T(D)\Big|_{D=e^{-\gamma_b R_c}} \tag{19}$$

The bit error probability P_b is more useful measure than the first-event error probability P_e to measure the performance of the convolution code. Transfer function can be also represented as

$$T(D,N) = \sum_{d=d_{\text{free}}}^{\infty} a_d D^d N^{f(d)}$$

where exponent in the factor N denotes the number of information bit errors (i.e. number of 1s). Here $f(d)$ denote the exponent of N as a function of d. The average bit error probability is upper-bounded by multiplying each pairwise error probability $P_2(d)$ by the corresponding number of incorrectly decoded information bits, for each possible incorrect path that merges with the correct path at the node B.

Taking the derrivative of T(D,N) with respect to N and setting $N = 1$, we get

$$\left.\frac{dT(D,N)}{dN}\right|_{N=1} = \sum_{d=d_{\text{free}}}^{\infty} a_d f(d) D^d = \sum_{d=d_{\text{free}}}^{\infty} \beta_d D^d$$

where $\beta_d = a_d f(d)$. Thus the bit error probability for k = 1 is upperbounded by

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d P_2(d) \tag{20}$$

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d Q\left(\sqrt{2\gamma_b R_c d}\right) \tag{21}$$

Here, we can write it as follows from the derivations we did in previous pages,

$$Q\left(\sqrt{2\gamma_b R_c d}\right) \leq e^{-\gamma_b R_c d} = \left. D^d \right|_{D=e^{-\gamma_b R_c}} \tag{22}$$

and thus, the bit error probability can be written as,

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d e^{-\gamma_b R_c d} \tag{23}$$

This formula derived above is the absolute worst case bound for BER, but in our following part of comparison of theoretical and simulated results for rate half convolutional code, we have taken a looser upper bound (truncated union bound) which is tighter than this and more practical. (The formula of truncated union bound will be derived from derivative of transfer function at N=1 in section 4.)

3 Probability of error in hard decoding

We compute the Hamming distance for hard decision decoding between the received codeword and all other 2k possible codewords. Then we select the codeword that is closest to the codeword received. For simplicity we assume that all zero codeword is transmitted.

We will begin by determining the first-event error probability. Suppose that the path having the distance d is compared with the all-zero at some node B.

From the error correcting capacity $tc = \lfloor \frac{d-1}{2} \rfloor$

If d is odd, then the all-zero path will be correctly selected if the number of errors in the receive sequence is less than $\frac{1}{2}(d+1)$, otherwise the incorrect path will be selected. The probability of selecting the incorrect path is

$$P_2(d) = \sum_{k=\lceil \frac{d+1}{2} \rceil}^{d} \binom{d}{k} p^k (1-p)^{d-k} \tag{24}$$

Where p is the probability of a bit error for the $BSC(p)$ channel.

If d is even, the incorrect path is selected when the number of errors exceed $\frac{1}{2}d$. If the number of errors equals $\frac{1}{2}d$, there is the tie between the two paths (means both the paths are having same hamming distance with the received sequence), which may be resolved by randomly selecting one of the two paths. Thus the probability of selecting incorrect path is

$$P_2(d) = \sum_{k=\lceil d/2+1 \rceil}^{d} \binom{d}{k} p^k (1-p)^{d-k} + \frac{1}{2}\binom{d}{d/2} p^{d/2}(1-p)^{d/2} \tag{25}$$

There will be many paths with different distances that merge with the all-zero path at a given node. Due to the similar reason of the non-disjoint paths as we considered in the case of soft decision decoding, we can upper bound this error probability by the sum of the pairwise error probabilities $P_2 d$ over all possible paths that merge with the all-zero path at the given node.

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d P_2(d) \tag{26}$$

where a_d represents the number of paths corresponding to the distance d.

Now using the upper-bound on Equation (22)

$$\begin{aligned}
P_2(d) &= \sum_{k=\frac{(d+1)}{2}}^{d} \binom{d}{k} p^k (1-p)^{d-k} \\
&\leq \sum_{k=\frac{(d+1)}{2}}^{d} \binom{d}{k} p^{d/2}(1-p)^{d/2} \\
&= p^{d/2}(1-p)^{d/2} \sum_{k=\frac{d+1}{2}}^{d} \binom{d}{k} \\
&\leq p^{d/2}(1-p)^{d/2} \sum_{k=0}^{d} \binom{d}{k} \\
&\leq 2^d p^{d/2}(1-p)^{d/2} \\
\therefore P_2(d) &< [4p(1-p)]^{d/2}
\end{aligned} \tag{27}$$

Similarly we can prove for Equation (23).

Use Equation (25) in the Equation (24)

$$P_e < \sum_{d=d_{\text{free}}}^{\infty} a_d [4p(1-p)]^{d/2} \tag{28}$$

$$P_e < T(D)|_{D=\sqrt{4p(1-p)}}$$

The measure of the performance of the convolution code is given by bit error probability. As we have done in the soft decision decoding, from Equation (20), we can write the expression for the upper bound on the bit error probability for $k = 1$, in the form

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d P_2(d)$$

where $\beta_d = a_d f(d)$. Now, we can write,

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d [4p(1-p)]^{d/2} \tag{29}$$

The upper bound on P_b can be also expressed as

$$P_b < \left. \frac{dT(D,N)}{dN} \right|_{N=1,\, D=\sqrt{4p(1-p)}} \tag{30}$$

4 Theoretical Outcomes of BER

4.1 Soft Decision Decoding:

We will discuss everything in the context of rate 1/2 convolutional code.

As derived previously, the bit error rate for soft viterbi decoding can be written as

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d Q\left(\sqrt{2\gamma_b R_c d}\right)$$

Now, here β_d consist of the weights of ones in the particular path with distance d. We can rewrite the above equation like follows, which is reduced in terms of derivative of the transfer function with respect to N

$$P_b < \left. \sum_{d=d_{\text{free}}}^{\infty} \beta_d D^d \right|_{D=e^{-\gamma_b R_c}}$$

Here we know that,

$$\begin{aligned} T(D,N,J) &= \frac{D^5 J^3 N}{1 - DJN(1+J)} \\ &= D^5 J^3 N + D^6 J^4 N^2 (1+J) + ... \end{aligned}$$

and

$$T(D,N) = D^5 N + 2D^6 N^2 + 4D^7 N^3 + ...$$

$$\frac{dT(D,N)}{dN} = D^5 + 4D^6N + 12D^7N^2 + ...$$

$$\left.\frac{dT(D,N)}{dN}\right|_{N=1} = D^5 + 4D^6 + 12D^7 + 32D^8 + ...$$

The above written formula is called the truncated transfer function, which is a good method to use for further analysis. Remember, we have written something like this previously,

$$T(D,N) = \sum_{d=d_{\text{free}}}^{\infty} a_d D^d N^{f(d)}$$

We can see that the truncated transfer function is just of the form,

$$\left.\frac{dT(D,N)}{dN}\right|_{N=1} = \sum_{d=d_{\text{free}}}^{\infty} a_d f(d) D^d$$

where $\beta_d = a_d f(d)$. Now, we know that $D^d = e^{-\gamma_b R_c d}$ thus,

$$\left.\frac{dT(D,N)}{dN}\right|_{N=1, D=e^{-\gamma_b R_c}} = \sum_{d=d_{\text{free}}}^{\infty} \beta_d e^{-\gamma_b R_c d} > P_b$$

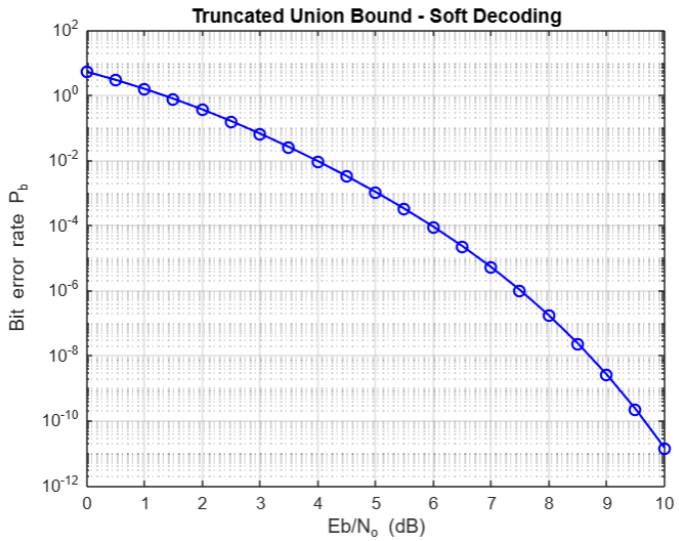


Figure 4: Theoretical BER of soft decoding for kc = 3, r = (1/2)

So, this is how the truncated union bound is achieved. Now, for the code that we have done, we have considered only the first few terms of this formula which are dominant in the series of summation. That is

$$P_b < \sum_{d=d_{\text{free}}}^{d_{\text{free}}+6} \beta_d e^{-\gamma_b R_c d}$$

Because, we haven't considered the whole infinite progressive series, all contributing terms aren't there and thus this cannot be called as worst case union bound, it is tighter than that and more practical.

We achieved the graph as Figure 4 from this.

4.2 Hard Decision Decoding:

In the previous section of the hard viterbi decoding, we had derived the formula like below,

$$P_2(d) = \sum_{k=\frac{d+1}{2}}^{d} \binom{d}{k} p^k (1-p)^{d-k}$$

and further,

$$P_2(d) < [4p(1-p)]^{d/2}$$

$$P_b < \sum_{d=d_{\text{free}}}^{\infty} \beta_d P_2(d)$$

which is equivalent to,

$$P_b < \left. \frac{dT(D,N)}{dN} \right|_{N=1,\, D=\sqrt{4p(1-p)}}$$

Now, we will again look at this both and derive our formula to use to find the bounded BER for hard viterbi decoding. Here, we consider the below formula to use in derivations for a better and tighter approximation,

$$P_2(d) = \sum_{k=\frac{d+1}{2}}^{d} \binom{d}{k} p^k (1-p)^{d-k}$$

As we have written here, the probability 'p' here is the approximation of bit error probability as a BSC(p) channel with bit flip probability 'p'. In our case, we know that it is the probability where we decode the bit incorrectly and that is,

$$p = Q\left(\sqrt{2\gamma_b R_c}\right)$$

and therefore,

$$P_2(d) = \sum_{k=\frac{d+1}{2}}^{d} \binom{d}{k} (Q(\sqrt{2\gamma_b R_c}))^k (1 - Q(\sqrt{2\gamma_b R_c}))^{d-k}$$

Now, we can again map this into the truncated union bound as we did in soft decision decoding part where we considered only some first fewer terms of the truncated transfer function to approximate the BER of hard viterbi decoding tighter and practical than worst case union bound. Thus, the BER here will be,

$$P_b < \sum_{d=d_{\text{free}}}^{d_{\text{free}}+6} \beta_d P_2(d)$$

Below is the BER curve that we will get by the derived formula,

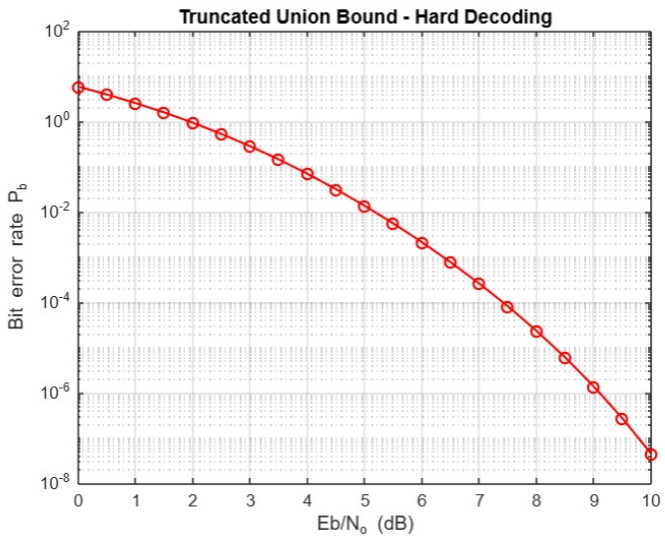


Figure 5: Theoretical BER of hard decoding for kc = 3, r = (1/2)

5 Comparison of Simulated and Theoretical results

Below is the combined BER curves of theoretical and simulated soft and hard viterbi decoding for convolutional code with code rate = 1/2 an constraint length 3. Further comparison and analysis is given as follows:

5.1 Simulated Hard vs Theoretical Hard

- Both are exponentially decaying because of the high SNR.
- Simulated BER is slightly worse than the theoretical BER due to practical decoding limitations and finite-length effects.
- At higher E_b/N_0 (> 6 dB), the simulated curve closely approaches the theoretical bound, which validates the accuracy of the viterbi algorithm under hard

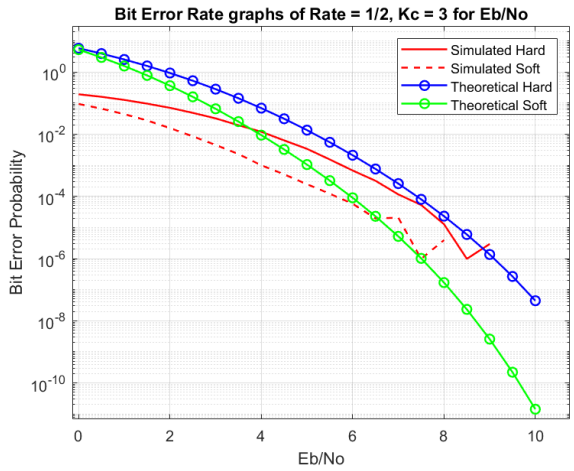


Figure 6: Combined BER of soft and hard decoding theoretical and simulated

decision.
- For E_b/N_0 around 5–7 dB, theoretical BER is approximately 10^{-2} to 10^{-4}, which the simulation closely follows.
- Truncated union bound gives a close estimate of actual BER performance especially at high SNRs.
- Thus, truncated union bound utility shows good agreement and approximation with simulated BER.

5.2 Simulated Soft vs Theoretical Soft

- Soft decoding shows a clear advantage (1.5–2 dB gain) over hard decoding, both in simulation and theory.
- Simulated and theoretical BER curves align closely, especially in the mid to high E_b/N_0 range.
- Theoretical soft decoding can reach extremely low BERs (upto 10^{-12}) beyond $E_b/N_0 = 9$ dB, which the simulation approaches but does not exactly reach due to practical limits.
- The soft decision decoding curves are steeper, that indicates better error correction.
- Here also truncated union bound gives a good and closer approximation to the simulated results.
- Thus, agreement between simulated and theoretical results validates the reliability on simulated performance.

CT216 - Group - 19 - Convolutional Codes

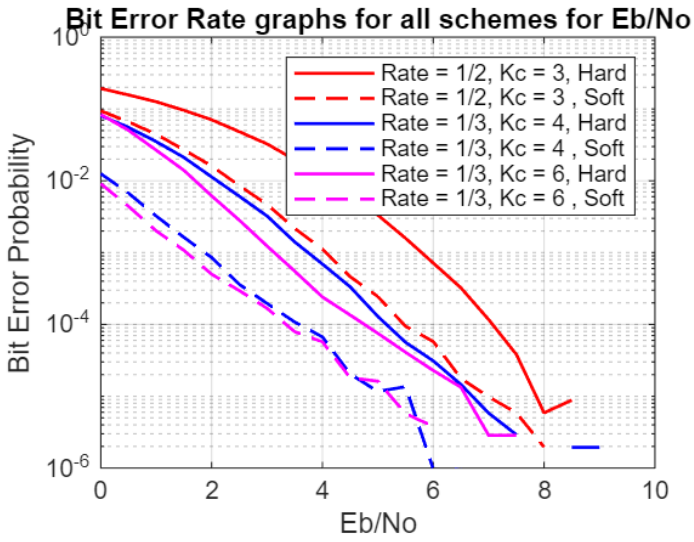
Simulation & Outputs:

```
tic;
Nsim = 10000;
input_size = 100; % Input message size
Kc = [3 4 6]; % Constraint lengths list
R = [1/2 1/3 1/3]; % Code rates
generatorOctal = {[5 7]; [13 15 17]; [47 53 75]};
EbNo_dB = 0:0.5:10;
EbNo_lin = 10.^(EbNo_dB/10);

% State Transition data tables
State_Table_1 = State_Table_generator(Kc(1),R(1), generatorOctal{1,:});
State_Table_2 = State_Table_generator(Kc(2),R(2), generatorOctal{2,:});
State_Table_3 = State_Table_generator(Kc(3),R(3), generatorOctal{3,:});

prob_decoding_failure_hard = zeros(3, length(EbNo_lin));
prob_decoding_failure_soft = zeros(3, length(EbNo_lin));
BER_hard = zeros(3, length(EbNo_lin));
BER_soft = zeros(3, length(EbNo_lin));

for c = 1 : length(Kc)
    for i = 1 : length(EbNo_lin)
        failures_hard = 0;
        failures_soft = 0;
        bits_in_error_hard = 0;
        bits_in_error_soft = 0;
        total_error_bits_hard = 0;
        total_error_bits_soft = 0;

        for j = 1: Nsim
            % Input message
            message = rand(1,input_size) < 0.5;
            message = [message, zeros(1,Kc(c) - 1)];

            % Encoding
            codeword = Convolutional_Encoder(message, generatorOctal{c,:}, R(c),
Kc(c));

            % Modulation
            modulated = 1 - 2*codeword;

            % Corruption
            noise_power = 1/(EbNo_lin(i));
            sig = sqrt(noise_power);
            % AWGN definition
```

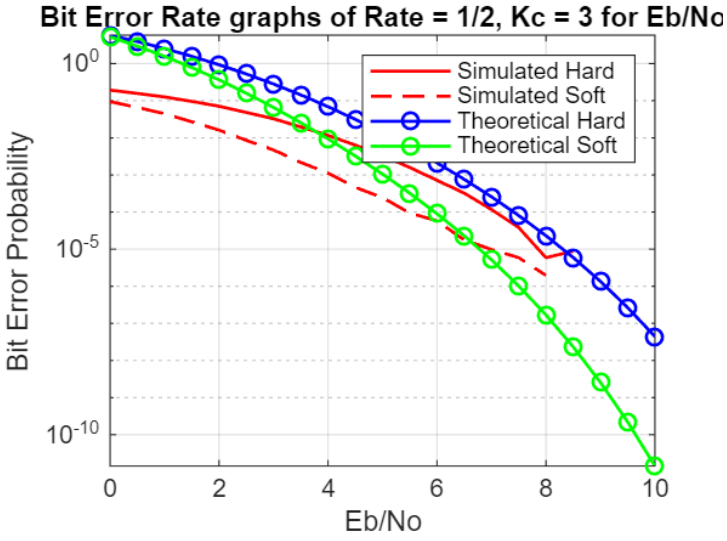
```
            noise = sig .* randn(1,length(modulated));

            % Corruption
            corrupted = modulated + noise;

            % Demodulation
            demodulated = corrupted <0;

            switch c
                case 1
                    State_Table = State_Table_1;
                case 2
                    State_Table = State_Table_2;
                otherwise
                    State_Table = State_Table_3;

            end

            decoded_mssg_hard = Convolutional_Decoder(demodulated, Kc(c), R(c),
'H', State_Table);
            decoded_mssg_soft = Convolutional_Decoder(corrupted, Kc(c), R(c), 'S',
State_Table);

            % Remaining number of errors in decoded message
            bits_in_error_hard = sum(bitxor(decoded_mssg_hard, message));
            bits_in_error_soft = sum(bitxor(decoded_mssg_soft, message));

            % Total bits in error over a simula
            total_error_bits_hard = total_error_bits_hard + bits_in_error_hard;
            total_error_bits_soft = total_error_bits_soft + bits_in_error_soft;

            if bits_in_error_hard > 0
                failures_hard = failures_hard + 1;
            end
            if bits_in_error_soft > 0
                failures_soft = failures_soft + 1;
            end
        end
        prob_decoding_failure_hard(c,i) = failures_hard/Nsim;
        prob_decoding_failure_soft(c,i) = failures_soft/Nsim;

        BER_hard(c,i) = total_error_bits_hard/(Nsim * length(decoded_mssg_hard));
        BER_soft(c,i) = total_error_bits_soft/(Nsim * length(decoded_mssg_soft));
    end
end

% For Rate = 1/2 and Kc = 3, Theoretical BER is calculated below

% Calculate theoretical BER for hard viterbi decoding
BER_Theoretical_Hard= zeros(size(EbNo_lin));
```

```
% Approximation as a BSC channel with bit flip probability 'p'
p = qfunc(sqrt(2 * R(1) * EbNo_lin));
d_vals = 5:11; %Some first dominant terms of transfer function in consideration

% A_d = [1, 2, 4, 8, 16, 32 ,....] --> We will consider first dominant terms
% f(d) = [1, 2, 3, 4, 5, 6,....] --> power of N in transfer function

beta_d = [1, 4, 12, 32, 80, 192, 448]; % (A_d x f(d)) --> weights of D^d in DT/DN
for j = 1:length(EbNo_lin)
    Pb = 0;
    for idx = 1:length(d_vals)
        d = d_vals(idx);
        sum_prob = 0;
        for i = ceil(d/2):d
            % Central formula
            sum_prob = sum_prob + nchoosek(d, i) * p(j)^i * (1 - p(j))^(d - i);
        end
        Pb = Pb + beta_d(idx) * sum_prob;
    end
    BER_Theoretical_Hard(j) = Pb;
end

% Calculate theoretical BER for soft viterbi decoding
BER_Theoretical_Soft = zeros(size(EbNo_lin));

for j = 1:length(EbNo_lin)
    D = 0;
    for i = 1:length(d_vals)
        % Central formula
        D = D + beta_d(i) * exp(-d_vals(i) * EbNo_lin(j) * R(1));
    end
    BER_Theoretical_Soft(j) = D;
end


% Bit error rates
figure;
semilogy(EbNo_dB, BER_hard(1,:), 'r-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, BER_soft(1,:), 'r--', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_hard(2,:), 'b-', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_soft(2,:), 'b--', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_hard(3,:), 'm-', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_soft(3,:), 'm--', 'LineWidth', 1.2);
title('Bit Error Rate graphs for all schemes for Eb/No');
legend('Rate = 1/2, Kc = 3, Hard', 'Rate = 1/2, Kc = 3 , Soft', ...
    'Rate = 1/3, Kc = 4, Hard', 'Rate = 1/3, Kc = 4 , Soft', ...
    'Rate = 1/3, Kc = 6, Hard', 'Rate = 1/3, Kc = 6 , Soft');
xlabel('Eb/No');
```

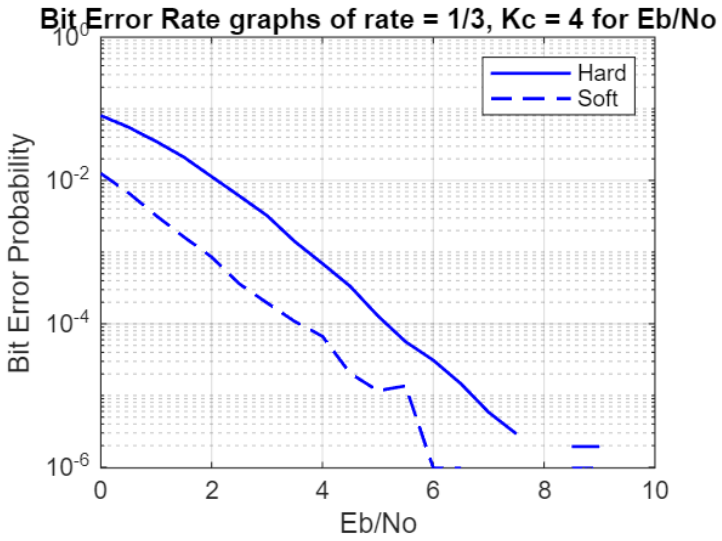
```
ylabel('Bit Error Probability');
xlim([min(EbNo_dB) max(EbNo_dB)]);
ylim([1e-6 1e0]);
grid on;
```



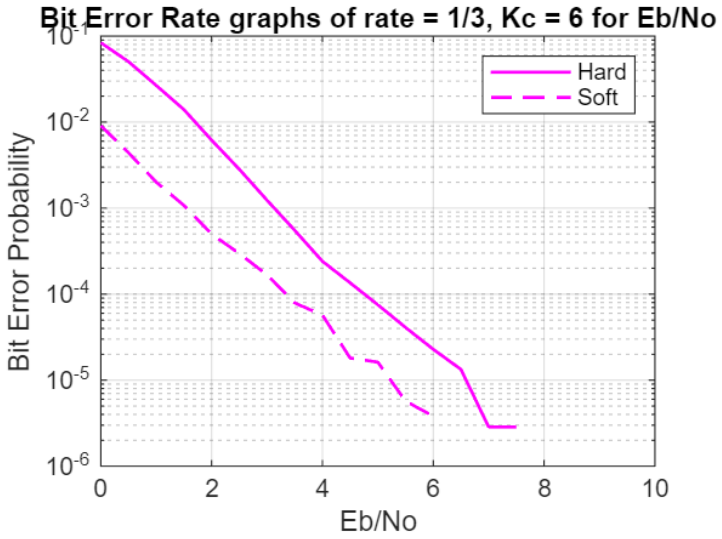

```
figure;
semilogy(EbNo_dB, BER_hard(1,:), 'r-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, BER_soft(1,:), 'r--', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_Theoretical_Hard, 'b-o', 'LineWidth', 1.2);
semilogy(EbNo_dB, BER_Theoretical_Soft, 'g-o', 'LineWidth', 1.2);
title('Bit Error Rate graphs of Rate = 1/2, Kc = 3 for Eb/No');
legend('Simulated Hard', 'Simulated Soft', 'Theoretical Hard', 'Theoretical Soft');
xlabel('Eb/No');
ylabel('Bit Error Probability');
xlim([min(EbNo_dB) max(EbNo_dB)]);
grid on;
```



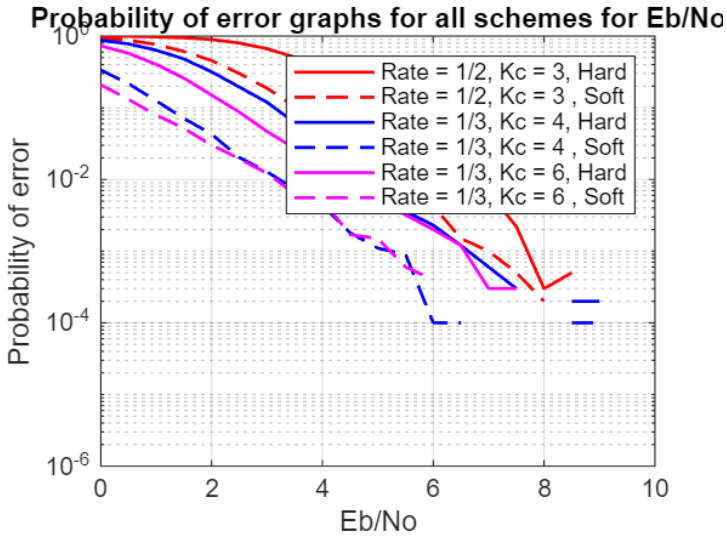
```
figure;
semilogy(EbNo_dB, BER_hard(2,:), 'b-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, BER_soft(2,:), 'b--', 'LineWidth', 1.2);
title('Bit Error Rate graphs of rate = 1/3, Kc = 4 for Eb/No');
legend('Hard', 'Soft');
xlabel('Eb/No');
ylabel('Bit Error Probability');
xlim([min(EbNo_dB) max(EbNo_dB)]);
grid on;
```



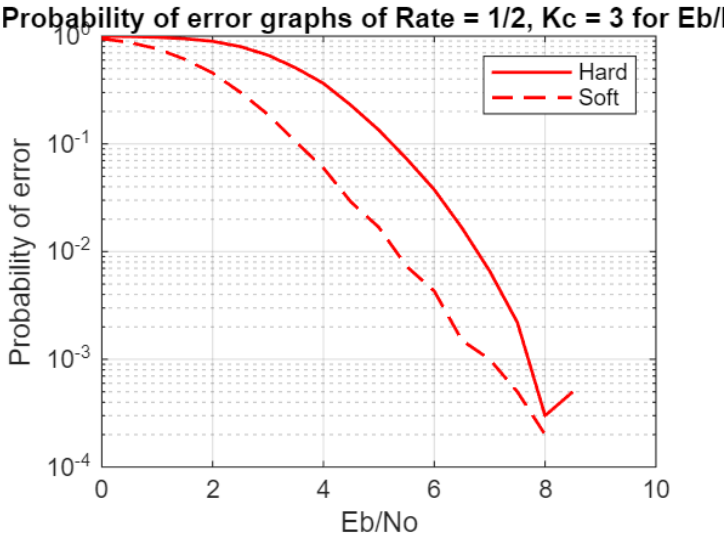

```
figure;
semilogy(EbNo_dB, BER_hard(3,:), 'm-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, BER_soft(3,:), 'm--', 'LineWidth', 1.2);
title('Bit Error Rate graphs of rate = 1/3, Kc = 6 for Eb/No');
legend('Hard', 'Soft');
xlabel('Eb/No');
ylabel('Bit Error Probability');
xlim([min(EbNo_dB) max(EbNo_dB)]);
grid on;
```




```
% Probability of decoding failure
figure;
semilogy(EbNo_dB, prob_decoding_failure_hard(1,:), 'r-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, prob_decoding_failure_soft(1,:), 'r--', 'LineWidth', 1.2);
semilogy(EbNo_dB, prob_decoding_failure_hard(2,:), 'b-', 'LineWidth', 1.2);
semilogy(EbNo_dB, prob_decoding_failure_soft(2,:), 'b--', 'LineWidth', 1.2);
semilogy(EbNo_dB, prob_decoding_failure_hard(3,:), 'm-', 'LineWidth', 1.2);
semilogy(EbNo_dB, prob_decoding_failure_soft(3,:), 'm--', 'LineWidth', 1.2);
title('Probability of error graphs for all schemes for Eb/No');
legend('Rate = 1/2, Kc = 3, Hard', 'Rate = 1/2, Kc = 3 , Soft', ...
    'Rate = 1/3, Kc = 4, Hard', 'Rate = 1/3, Kc = 4 , Soft', ...
    'Rate = 1/3, Kc = 6, Hard', 'Rate = 1/3, Kc = 6 , Soft');
xlabel('Eb/No');
ylabel('Probability of error');
xlim([min(EbNo_dB) max(EbNo_dB)]);
ylim([1e-6 1e0]);
grid on;
```

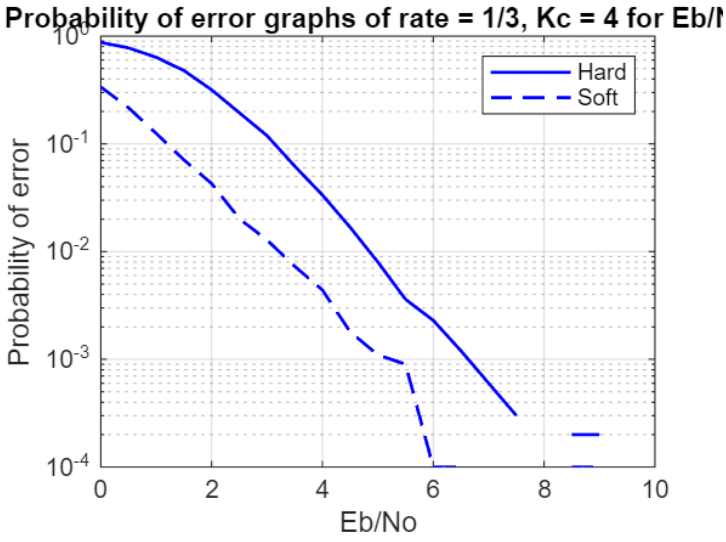



```
figure;
semilogy(EbNo_dB, prob_decoding_failure_hard(1,:), 'r-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, prob_decoding_failure_soft(1,:), 'r--', 'LineWidth', 1.2);
title('Probability of error graphs of Rate = 1/2, Kc = 3 for Eb/No');
legend('Hard', 'Soft');
xlabel('Eb/No');
ylabel('Probability of error');
xlim([min(EbNo_dB) max(EbNo_dB)]);
grid on;
```

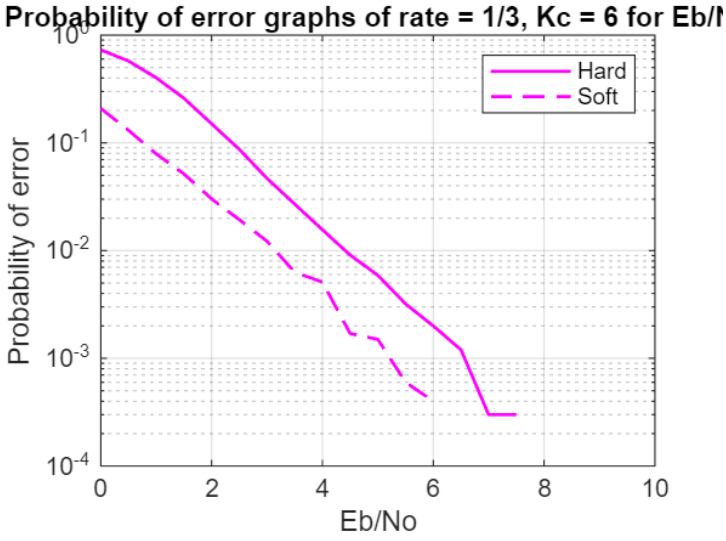



```
figure;
semilogy(EbNo_dB, prob_decoding_failure_hard(2,:), 'b-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, prob_decoding_failure_soft(2,:), 'b--', 'LineWidth', 1.2);
title('Probability of error graphs of rate = 1/3, Kc = 4 for Eb/No');
```

```
legend('Hard', 'Soft');
xlabel('Eb/No');
ylabel('Probability of error');
xlim([min(EbNo_dB) max(EbNo_dB)]);
grid on;
```




```
figure;
semilogy(EbNo_dB, prob_decoding_failure_hard(3,:), 'm-', 'LineWidth', 1.2);
hold on;
semilogy(EbNo_dB, prob_decoding_failure_soft(3,:), 'm--', 'LineWidth', 1.2);
title('Probability of error graphs of rate = 1/3, Kc = 6 for Eb/No');
legend('Hard', 'Soft');
xlabel('Eb/No');
ylabel('Probability of error');
xlim([min(EbNo_dB) max(EbNo_dB)]);

grid on;
```



```
toc;
```

```
Elapsed time is 17933.218701 seconds.
```

```

```

Functions:

Octal to Binary generator function for generator vector: -

```
function G = octal2bin_for_generators(a)
for i = 1:length(a)
    digitarray = dec2bin(oct2dec(a(i))) - '0';
    G(:,i) = digitarray';
end
end
```

Encoder Function: -

```
function enc = Convolutional_Encoder(mssg, gen, r, Kc)

N = length(mssg);
parse = zeros(1,Kc-1); % Extra Kc-1 zeros to handle initial empty regeister states
mssg = [parse,mssg];
idx = 1:N;

% The matrix that will be multiplied with generators directly
% Matrix whose each column has input and current state involved at a time
bits = mssg(idx' + (Kc-1:-1:0))';

% Deriving generators into binary vectors from octal values
G = octal2bin_for_generators(gen);

% Encoding
% multiplication of each state with generators and formation of output
v = mod(G.' * bits, 2);

% Encoded massage
opt = reshape(v, 1, []);
enc = opt;
end
```

State Table generator function to store the state data: -

```
%function for generating state_table for a given constraint length kc and
%code rate r
```

```
function t = State_Table_generator(kc, r, gen)

%generating generator matrix G from given decimal value of genrator
G = octal2bin_for_generators(gen);
k = kc-1;
ip = zeros(kc,1);
queue = [1]; %queue to maintain the order of the evaluation
t = cell(2^kc,4);
l = 1;
visited = zeros(1,2^k);
visited(1) = 1;

%this loop will generate the state table for every state and store
%curr_state, input, next state and output of the current state
while nnz(queue) ~= 0
    front = queue(1);
    queue(1) = [];

    vec = dec2bin(front-1,k);   %converting state number to binary of length k
    vec = vec-'0';  %convert string to numeric array
    vec = vec'; %convert row to column vector

    %for each state we have two ip : 0/1
    for j = 0:1
        ip = [j;vec];

        t{l,1} = front;   %current_state
        t{l,2} = j;       %input_bit

        nextstr = num2str((ip(1:end-1))'); % next_state
        nextstr = nextstr(~isspace(nextstr));

        val=bin2dec(nextstr); %convert state to decimal
        val=val+1;

        % We visit each node twice one time for input 0 and one time for input 1
        % Again adding to q if node not visited twice
        if(visited(val)<1)
            queue(end+1) = val;
            visited(val) = visited(val)+1;
        end

        t{l,3} = val;     % Next state in decimal
        v = mod(G.'*ip, 2);  % Output v = Generator_matrix * input
        out = num2str(v);    % store output in string
        out = out(~isspace(out));
        t{l,4} = out;       % Output

        if j == 0
```

```
            l = l+2;    % Storing states at every other index for easier access
later
        end
    end

    % Moving to index 2 to fill the table again after reaching last to second index
    if l == 2^kc-1
        l = 2;
    else
        l = l + 2;
    end

end

i = 1;
l = 1;

%Rename states to make backtracking easier later
for p = 1:2^kc
    if mod(p,2)==0
        continue;
    end

    t{p,1} = i;
    t{p,3} = l;
    t{p+1,1} = i + 2^k/2;
    t{p+1,3} = l;
    l = l + 1;

    if mod(l,2) ~= 0
        i = i + 1;
    end
end

end
```

Decoder Function:

```
function [mssg] = Convolutional_Decoder(ip, kc, r, H_or_S, state_data)
k= kc-1;
nop = 1/r;
s = length(ip);
ip = reshape(ip,[], s/nop);

prev_metric = zeros(1,2^k); % Latest updates cost metric of each node

% storing cost metric, ipbit and parent stage of each node at each level,
% 3D matrix containing all the important data of trellis
```

```
data = zeros(s/nop, 3, 2^k);

% DP and tabulation method for cost computation
for i = 1:s/nop
    lim = min(2^i, 2^kc);
    l=1;
    tmpstr = zeros(2^k, 3);
    for j = 1:lim

        % Output chunk for comparison with revieved output chunk
        arr = (state_data{l,4}' - '0');

        % Calculation of metric for particular state node
        if H_or_S == 'H'
            metric = sum(bitxor(arr, ip(:,i)'));
        else
            metric = (norm(ip(:,i)' - (1 - 2.*arr)))^2;
        end

        % Data metric calculation and updation
        if nnz(tmpstr(state_data{l,3},3))>0 && (metric +
prev_metric(state_data{l,1})>tmpstr(state_data{l,3},1))
            data(i,:,state_data{l,3}) = tmpstr(state_data{l,3},:);
        else
            data(i,1,state_data{l,3}) = metric + prev_metric(state_data{l,1}); %
metric at current node
            data(i,2,state_data{l,3}) = state_data{l,2}; % prev input that drives
to current node
            data(i,3,state_data{l,3}) = state_data{l,1}; % parent state
            tmpstr(state_data{l,3},:) = data(i,:, state_data{l,3});
        end

        % Smart indexing for levelwise lookup in state data
        if lim==2^kc
            l = l + 1;
        else
            l = l + 2;
        end
    end
    prev_metric(:) = data(i,1,:);

end

% Backtracking and extraction of input message
it = size(data, 1);
decoded = zeros(1, s/nop);
[~, D] = min(data(end, 1, :));
parent = data(end, :, D);
while it~=0
    decoded(it) = parent(2);
```

```
    it = it - 1;
    if it==0
        break;
    end
    parent = data(it, :, parent(3));
end

% Decoded massage
mssg = decoded(1 : s/nop);
end
```