You (and/or your team; maximum of three students per team) are expected to produce a computer program to implement the Viterbi decoding algorithm for the *Voyager* code, i.e., the (2, 1, 6, 10) binary convolutional code with generator matrix

\[(G_1(D), G_2(D)) = (1 + D^2 + D^3 + D^5 + D^6, 1 + D + D^2 + D^3 + D^6).\]

**The Deliverable, Part 1.** I want you to run simulations with your Viterbi decoder to produce graphs showing the (approximate) relationship between \(E_b/N_0\) and the decoded bit error probability for the given convolutional code, for \(E_b/N_0\) ranging from 1 dB to 6dB, in increments of 0.5 dB. Graph 1 will show the results for “hard decision” decoding and Graph 2 will show the results for “soft decision” decoding. For comparison, also plot the uncoded bit error probability, given by the formula

\[P_{\text{uncoded}} = Q(\sqrt{2E_b/N_0}),\]

where

\[Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-t^2/2} dt.\]

**The Deliverable, Part II.** *This part is to entertain the Professor. Provided you turn in an answer, it will not affect your grade.* Select a random 4-digit seed equal to the last four digits of the SSN of the oldest member of your team. Using this seed, decode 100000 bits with a noise variance of exactly

\[\sigma = \ln 2,\]

with both hard and soft decision decoding. *Report the number of bit errors made by your decoder.*

**Important Fact:** For a binary code of rate \(R\) on the AWGN channel, the relationship between \(E_b/N_0\), the bit signal-to-noise ratio and \(\sigma^2\), the Gaussian noise variance, is given by

\[\sigma^2 = \left(2R \frac{E_b}{N_0}\right)^{-1} ,\]

so for example for a \(R = 1/2\) code like the *Voyager* code, the relationship is simply

\[\sigma^2 = \left( \frac{E_b}{N_0} \right)^{-1} .\]

Finally remember that \(E_b/N_0\) is normally quoted in “dBs,” where a dimensionless quantity \(x\) equals \(10 \log_{10} x\) dB’s. Thus for example, a value of \(E_b/N_0\) of 3.5 dB for the Voyager code corresponds to a value of \(\sigma^2 = 0.4467.\)
Additional details on Viterbi Decoder Project.

1. Use the recursion

\[ p_{n+6} = p_{n+1} \oplus p_n \quad \text{for } n \geq 0 \]

with the initial conditions

\[ p_0 = 1, p_1 = p_2 = p_3 = p_4 = p_5 = 0, \]

to generate the \( N \) information bits. Ensure that the generated sequence is 10000100001\ldots and is periodic with period 63.

2. Encode the information sequence using the generator polynomials \( G_1(D) \) and \( G_2(D) \) given above.

3. The encoder outputs 0’s and 1’s. However, the input to the AWGN is \( \pm 1 \). Therefore, map 0’s to \(+1\)’s and 1’s to \(-1\)’s.

4. To simulate the AWGN with soft decision decoding, add the mean zero, variance \( \sigma^2 \) normal (Gaussian) random variables generated by the following segment of pseudo-code, to the \( \pm 1\)’s generated at the previous step. This program outputs two random variables, \( n_1 \) and \( n_2 \). Use \( n_1 \) (resp. \( n_2 \)) for the encoder output corresponding to the generator polynomial \( G_1(D) \) (resp. \( G_2(D) \)). \( \text{SEED} \) and \( \sigma \) (i.e., \( E_b/N_0 \)) will be specified at the time of testing your program. \text{urand()} is a function which generates a random variable uniformly distributed in the interval \([0,1]\).

\begin{verbatim}
main()
{
...
global iurv;
...
iurv = SEED;
...
}

normal(n1, n2, \sigma) /* See “Donald E.Knuth, The Art of Computer Programming, Vol.2, p.104” */
{
    do {
        \( x_1 = \text{urand}() \);
        \( x_2 = \text{urand}() \);
    }
}
\end{verbatim}
\[ x_1 = 2x_1 - 1; \]
\[ x_2 = 2x_2 - 1; \]
\[ /* x_1 \text{ and } x_2 \text{ are now uniformly distributed in } [-1, +1] */ \]
\[ s = x_1^2 + x_2^2; \]
\[ } \text{ while } (s \geq 1.0) \]
\[ n_1 = \sigma x_1 \sqrt{-2 \ln s/s}; \]
\[ n_2 = \sigma x_2 \sqrt{-2 \ln s/s}; \]
\]
\}
\]
urand()
{
\}

iurv = (14157iurv + 6925)(mod32768);
return iurv/32767;

5. For “hard decision” decoding:

(a) Take the sign of the output of the AWGN (Define \( \text{Sign(0)} = +1. \))

(b) Map +1’s to 0’s and -1’s to 1’s.

6. Truncate your survivors to length 32 and output the oldest bit on the survivor with the least metric (“Best State Decoding”). To \( \text{decode } N \text{ bits, generate } N + 32 \text{ bits in } (1). \)

Your program should output the fraction of decode bits in error (Bit Error Rate) in both cases.

The following table lists some typical values.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \sigma )</th>
<th>( E_b/N_0 )</th>
<th>( \text{SEED} )</th>
<th>( \text{BER (soft)} )</th>
<th>( \text{BER (hard)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.8</td>
<td>1.94dB</td>
<td>101</td>
<td>0.010</td>
<td>0.158</td>
</tr>
<tr>
<td>1000</td>
<td>0.9</td>
<td>0.92dB</td>
<td>111</td>
<td>0.107</td>
<td>0.225</td>
</tr>
</tbody>
</table>

7. What to do in case of tied metrics? There are two cases to consider. First, in the “add-compare-select” step the two metrics could be equal. In this case you can safely select either choice; it won’t affect the decoder performance. Second, in choosing the survivor with the best metric, if there are two or more best states, take the majority vote of the oldest bits on the winning states. If this is still tied, choose arbitrarily.

8. Notes on simulation accuracy. As \( E_b/N_0 \) increases, you will find that decoder bit errors become quite rare. I suggest, but do not insist on, the following approach. At a given value of \( E_b/N_0 \), run your decoder until you have seen at least \( K \) errors. If this required \( N \) decoded bits, your estimate of the bit error probability is \( K/N \). The larger \( K \), is the more
accurate your plot will be, but $K$ will be limited by you decoder’s speed. For example, with $E_b/N_0 = 4.0dB$, and sft decision decoding, the BER is around $10^{-5}$, which means you will need to decode around $10000K$ bits before $K$ bit errors have occurred.