Details of Class Project #1  
Due date: Week of May 7

You (and/or your team; maximum of four students per team) are expected to produce a computer program to implement the BCJR “APP” decoding algorithm (ideally, in “log” form) for the “Berrou” code, i.e., the rate 1/2, memory 4, systematic recursive binary convolutional code with generator matrix

\[(1, G_1(D)/G_2(D)) = (1, \frac{1 + D^4}{1 + D + D^2 + D^3 + D^4}),\]

with encoding circuit as shown in Figure 2b of Berrou’s paper. You are expected to implement the code in truncated form, with each codeword representing \(k = 1024\) information bits, plus the 4 dummy bts required to force the encoder to the all-zero state. (This makes the overall code a (2056, 1024) binary linear code.)

- **The primary goal** is for you to run simulations to produce a histogram of the decoder’s log-likelihood ratios \(LLR_1, \ldots, LLR_k\) for the information bits \(u_1, \ldots, u_k\), for 6 values of \(E_b/N_0\): 1 dB, 2 dB, \ldots, 6 dB. (Since the distribution of the LLR for a \(-1\) information bit will be the negative of that for a \(+1\) information bit, your histograms should correct for this bias. In other words, I want a histogram of \(u_i \cdot LLR_i\), for \(i = 1, \ldots, k\).)

- **The secondary goal** of the project is to produce a graph which shows the (approximate) relationship between \(E_b/N_0\) and the decoded bit error probability for the given code, for \(E_b/N_0\) ranging from 1 dB to 6dB, in increments of 1 dB. (To decode the \(i\)th information bit \(u_i\), you compute \(LLR_i\) using the BCJR algorithm and then make the decision

\[
\hat{u}_i = \begin{cases} 
+1 & \text{if } LLR_i \geq 0 \\
-1 & \text{if } LLR_i < 0.
\end{cases}
\]

- **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 Berrou code, the relationship is simply

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

Remember that \(E_b/N_0\) is always 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 3.0 dB for the Berrou code corresponds to a value of \(\sigma^2 = 0.5012\).
**Additional details on Class Project 1.**

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 \( k \) information bits. Ensure that the generated sequence is 10000100001... and is periodic with period 63.

2. Encode the information sequence using the generator matrix \((1, \frac{G_1(D)}{G_2(D)})\) given above. Refer to the encoder circuit in Figure 1(b) in the Berrou paper, if necessary.

3. The encoder outputs 0’s and 1’s. However, the input to the AWGN is ±1. Therefore, map 0’s to +1’s and 1’s to -1’s. Denote the ±1 input (information) stream by \( u_1, u_2, \ldots, u_k \), and the corresponding ±1 output stream by \((u_1, x_1), (u_2, x_2), \ldots (u_k, x_k)\).

4. To simulate the AWGN, add the mean zero, variance \( \sigma^2 \) normal (Gaussian) random variables generated by the following segment of pseudo-code, to the \((u_i, x_i)\)’s generated at the previous step. This program outputs two random variables, \( n_1 \) and \( n_2 \). Add \( n_1 \) to \( u_i \) and \( n_2 \) to \( x_i \). In your simulations, use a different value of \texttt{SEED} for each run. \texttt{urand()} is a function which generates a random variable uniformly distributed in the interval [0, 1].

```c
main()
{
  ...
  global iurv;
  ...
  iurv = SEED;
  ...
  ...
}

normal(n_1, n_2, \sigma) /* See “Donald E.Knuth, The Art of Computer Programming, Vol.2, p.104” */
{
  do {
    x_1 = urand();
    x_2 = urand();
  }
  ...
}
\[
\begin{align*}
x_1 &= 2x_1 - 1; \\
x_2 &= 2x_2 - 1; \\
\text{/* } x_1 \text{ and } x_2 \text{ are now uniformly distributed in } [-1,+1] */
\end{align*}
\]
\[
s = x_1^2 + x_2^2;
\]
\[
} \text{ while } (s \geq 1.0)
\]
\[
n_1 = \sigma x_1 \sqrt{-2 \ln s / s};
n_2 = \sigma x_2 \sqrt{-2 \ln s / s};
\]
\[
urand()
\]
\[
\{
\begin{align*}
iurv &= (14157iurv + 6925)(mod32768);
\text{return } iurv/32767;
\end{align*}
\}
\]
5. Implement the BCJR algorithm in “log” form, as discussed in class, using the approximation to \(\log(x + y)\) specified in the solutions to HW assignment 2. Thus
\[
\log(x + y) = \max(\log x, \log y) + f(|\log x - \log y|),
\]
where \(f(z)\) is an approximation to the function \(\log(1 + e^{-z})\). Use the branch metric \(\gamma = (x \cdot y)/\sigma^2\), where \(x = (x_1, x_2)\) is the two-bit branch label and \(y = (y_1, y_2)\) is the corresponding pair of received symbols.