EE/Ma 127c Error-Correcting Codes Ling Li
draft of May 11, 2001 `ling@cs.caltech.edu`

Project 1 – BCJR algorithm

This project report is organized into 3 parts: 1. some details of the implementation of the BCJR algorithm; 2. the result of the computer tests (the main result is in Figure 1); 3. other interesting things, such as the performance of the algorithm when σ^2 is not known precisely.

Let $u(t)$, $\boldsymbol{s}(t)$, $\boldsymbol{x}(t)$, and $\boldsymbol{y}(t)$ be the information bit, state vector, codebits, and the noisy codebits at time t. Assume that the total time is $T = k + 4$ ($k = 1024$ is the number of the information bits and 4 is the number of dummy bits). Let $\boldsymbol{U}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$ be the information, codeword, and noisy codeword from time 0 to $T - 1$.

1. *Encoder*. The generator matrix is

$$\left(1, \frac{G_1(D)}{G_2(D)}\right) = \left(1, \frac{1 + D^4}{1 + D + D^2 + D^3 + D^4}\right). \tag{1}$$

   Then (refer to [Berrou et al., 1993, Fig. 1(b)])

$$\begin{aligned} \boldsymbol{s}(t+1) &= \boldsymbol{s}(t)\mathcal{A} + u(t)\mathcal{B}, \\ \boldsymbol{x}(t) &= \boldsymbol{s}(t)\mathcal{C} + u(t)\mathcal{D}, \end{aligned} \tag{2}$$

   where

$$\mathcal{A} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \mathcal{B} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}, \mathcal{C} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \mathcal{D} = \begin{pmatrix} 1 & 1 \end{pmatrix}.$$

   We can verify that $G(D)$ is as given in (1). For a more convenient way for programming:

$$\begin{aligned} \boldsymbol{s}(t+1) &= (s_1(t) + s_2(t) + s_3(t) + s_4(t) + u(t), s_1(t), s_2(t), s_3(t)), \\ \boldsymbol{x}(t+1) &= (u(t), s_1(t) + s_2(t) + s_3(t) + u(t)). \end{aligned}$$

   After the encoding, $x_i(t)$ is transformed to $\{-1, +1\}$ by mapping $0 \mapsto +1$ and $1 \mapsto -1$.

2. *The trellis*. There are $2^4 = 16$ states in every stage of the trellis graph. From (2), two edges exit from each state; and also two edges enter each state, since $\mathcal{A}$ is invertible.

   We use $\boldsymbol{s} \xmapsto{u/\boldsymbol{x}} \boldsymbol{s}'$ to denote an edge in the trellis graph, starting from state $\boldsymbol{s}$, accompanied by information bit u and codebits $\boldsymbol{x}$, and ending at state $\boldsymbol{s}'$.

3. *The weights*. The evidence is $\mathcal{E} = \boldsymbol{Y} = \boldsymbol{y}(0) \ldots \boldsymbol{y}(T-1)$. What we want to calculate is

$$p(u(t) = a|\mathcal{E}) = \alpha \sum_{\boldsymbol{U}:u(t)=a} p(\boldsymbol{U})p(\boldsymbol{Y}|\boldsymbol{X}) = \alpha \sum_{\boldsymbol{U}:u(t)=a} \prod_{i=0}^{T-1} p(u(i))p(\boldsymbol{y}(i)|\boldsymbol{x}(i)),$$

   where the notation α is the same as in [McEliece et al., 1998]; or $\alpha = p(\boldsymbol{Y})^{-1}$. (Note that the channel is memoryless.) To apply the BCJR algorithm, we define the weight of an edge $e = \boldsymbol{s} \xmapsto{u/\boldsymbol{x}} \boldsymbol{s}'$ as

$$w(e) = \pi(u)p(\boldsymbol{y}|\boldsymbol{x}),$$

where $\pi(u)$ is the *a priori* probability that the information bit is u.

Since we will use the log-likelihood ratio (LLR), it is good to express the weight in log form. Let σ^2 be the Gaussian noise variance of the channel. Then

$$\begin{aligned}
\log w(e) &= \log \pi(u) + \log p(\boldsymbol{y}|\boldsymbol{x}) \\
&= \log \pi(u) - \frac{1}{2\sigma^2}\|\boldsymbol{y}-\boldsymbol{x}\|^2 - \log 2\pi\sigma^2 \\
&= \log \pi(u) - \frac{1}{2\sigma^2}\left(y_1^2 + x_1^2 - 2x_1y_1 + y_2^2 + x_2^2 - 2x_2y_2\right) - \log 2\pi\sigma^2.
\end{aligned}$$

Since $x_i \in \{-1,+1\}$, x_i^2 is a constant. We can omit constants which are the same for $u=0$ and $u=1$. Thus we can define the logarithmic weight for $e \in E_{t,t+1}$ as

$$\tilde{w}(e) = \log \pi(u) + \frac{x_1y_1(t) + x_2y_2(t)}{\sigma^2}. \tag{3}$$

If $\pi(0) = \pi(1) = \frac{1}{2}$, we can further simplify (3) to

$$\tilde{w}(e) = \frac{x_1y_1(t) + x_2y_2(t)}{\sigma^2}. \tag{4}$$

4. *α and β.* The matrix W_i used in the forward-backward algorithm is very sparse. To reduce the computational complexity, we should use (see [McEliece, 2001])

$$\begin{aligned}
\alpha_t(\boldsymbol{s}') &= \sum_{\substack{u=0,1 \\ e_u:\boldsymbol{s}_u \mapsto \boldsymbol{s}' \in E_{t-1,t}}} \alpha_{t-1}(\boldsymbol{s}_u) w(e_u), \\
\beta_t(\boldsymbol{s}) &= \sum_{\substack{u=0,1 \\ e_u:\boldsymbol{s} \mapsto \boldsymbol{s}'_u \in E_{t,t+1}}} \beta_{t+1}(\boldsymbol{s}'_u) w(e_u).
\end{aligned}$$

The logarithmic version is

$$\tilde{\alpha}_t(\boldsymbol{s}') = \log\left(e^{\tilde{\alpha}_{t-1}(\boldsymbol{s}_0)+\tilde{w}(e_0)} + e^{\tilde{\alpha}_{t-1}(\boldsymbol{s}_1)+\tilde{w}(e_1)}\right), \tag{5}$$

$$\tilde{\beta}_t(\boldsymbol{s}) = \log\left(e^{\tilde{\beta}_{t+1}(\boldsymbol{s}'_0)+\tilde{w}(e_0)} + e^{\tilde{\beta}_{t+1}(\boldsymbol{s}'_1)+\tilde{w}(e_1)}\right). \tag{6}$$

The initial values of $\tilde{\alpha}_0(\boldsymbol{s})$ and $\tilde{\beta}_T(\boldsymbol{s})$ are defined as follows:

$$\tilde{\alpha}_0(\boldsymbol{s}) = \tilde{\beta}_T(\boldsymbol{s}) = \begin{cases} 0, & \boldsymbol{s} = \boldsymbol{0}; \\ -\infty, & \text{otherwise.} \end{cases}$$

5. *Log-likelihood Ratio.* We have (assume $e = \boldsymbol{s} \mapsto \boldsymbol{s}'$)

$$\begin{aligned}
\log p(u(t)=0|\mathcal{E}) &= \log \sum_{e \in E^{(0)}_{t-1,t}} \alpha_{t-1}(\boldsymbol{s}) w(e) \beta_t(\boldsymbol{s}') = \log \sum_{e \in E^{(0)}_{t-1,t}} e^{\tilde{\alpha}_{t-1}(\boldsymbol{s})+\tilde{w}(e)+\tilde{\beta}_t(\boldsymbol{s}')}, \\
\log p(u(t)=1|\mathcal{E}) &= \log \sum_{e \in E^{(1)}_{t-1,t}} \alpha_{t-1}(\boldsymbol{s}) w(e) \beta_t(\boldsymbol{s}') = \log \sum_{e \in E^{(1)}_{t-1,t}} e^{\tilde{\alpha}_{t-1}(\boldsymbol{s})+\tilde{w}(e)+\tilde{\beta}_t(\boldsymbol{s}')},
\end{aligned}$$

and thus

$$LLR_t = \log \frac{p(u(t)=0|\mathcal{E})}{p(u(t)=1|\mathcal{E})} = \log p(u(t)=0|\mathcal{E}) - \log p(u(t)=1|\mathcal{E}).$$

6. *Approximation of* $\log(e^x + e^y)$. We are asked to use the logarithmic weights ((3) or (4)) to calculate LLR. This poses a problem with (5) and (6): how to calculate $\log(e^x + e^y)$ without doing log or exp. From Homework 2.2,

   $$\log(e^x + e^y) = \max\{x, y\} + f(|x - y|),$$

   where $f(\Delta) = \log\left(1 + e^{-\Delta}\right)$. Here we approximate $f(\Delta)$. I tried:

   - $f \equiv 0$. That is, use only $\max\{x, y\}$ to approximate $\log(e^x + e^y)$.
   - 2-bit approximation. I tried two methods in my solution to Homework 2.2.

   It turns out that the approximation in my solution 2.2(b) is the best among those three, and $f \equiv 0$ is almost as good as the approximation in the solution 2.2(a).

7. *Histogram and normalization.* We are asked to plot a histogram for $\{LLR_t\}_{t=0}^{k-1}$, or more precisely, the adjusted LLR, i.e., $\{u(t) \cdot LLR_t\}_{t=0}^{k-1}$. We do not care about the LLR for the dummy bits.

   If we divide the range of LLR_t into M bins, and count the number of LLR_t's in each bin, then we can plot the histogram. However, we can do better. We can make a probability density from the histogram if we do a *normalization* before plotting.

   Assume all the bins have the same width, w. Denote the number of LLR_t in bin i by c_i. Then the probability density of LLR_t in bin i is

   $$p_i = \frac{c_i}{w \cdot k},$$

   since p_i is proportional to c_i and the 'integrate' of p_i, $\sum_i wp_i = 1$. For r runs (making the histogram of LLR more accurate),

   $$p_i = \frac{c_i}{w \cdot k \cdot r}.$$

8. *Random number generator.* From [Press et al., 1992, Section 7.1], a linear congruential method for generating random numbers is not free of sequential correlation on successive calls. If the period is as small as 32768, the number of lines on which pairs of points lie in 2D space will be no greater than $\sqrt{32768}$, or 181. In this project, we will need about 10^7 ($\approx 2T \times 5000$ runs) random numbers. So I used the random number generator `ran1()` in [Press et al., 1992, Section 7.1]. However, my results from the computer tests didn't show much difference.

9. *Basic results.* The BCJR decoding algorithm was run 5000 times to determine the distribution of the adjusted LLR and the average BER (bit error rate). These tests were performed for several different values of E_b/N_0. The results are shown in Figure 1.

   If we conjecture that the distribution of the adjusted LLR is Gaussian $\mathcal{N}(\ell, \sigma^2)$, we can calculate the BER from the Gaussian distribution as

   $$BER_t = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\ell)^2}{2\sigma^2}} dx = \frac{1}{\sqrt{\pi}} \int_{\ell/\sqrt{2\sigma^2}}^{\infty} e^{-t^2} dt = \frac{1}{2} \operatorname{erfc}\left(\frac{\ell}{\sqrt{2\sigma^2}}\right), \tag{7}$$

   where $\operatorname{erfc}(t) = \frac{2}{\sqrt{\pi}} \int_t^{\infty} e^{-t^2} dt$ can be calculated by `erfc()` in Matlab or `Erfc[]` in Mathematica. The mean ℓ and variance σ^2 can be statistically calculated from the adjusted LLR

data. Although Figure 1(a) shows a good match of the real distribution of LLR with the Gaussian approximation, Figure 1(b) implies that they two are not the same.

Below I list the average number of errors (in k bits) over 5000 runs of the decoding, and $k \cdot BER_t$, the number of errors from the corresponding Gaussian distribution.

E_b/N_0 (dB)	6	5	4	3	2	1
Actual error	0.0010	0.0224	0.2458	1.8120	9.3418	34.4954
Gaussian error	0.000144	0.00519	0.1029	1.1990	8.9720	39.0258

10. *Other interesting things.*

(a) *Random codeword.* The codeword we used in the previous tests was generated by the recursion $u_{n+6} = u_{n+1} \oplus u_n$ with period 63. I also tried randomly generated codewords. The results are similar.

E_b/N_0 (dB)	6	5	4	3	2	1
Actual error	0.0014	0.0246	0.2410	1.7946	9.3622	34.1514
Gaussian error	0.000147	0.00522	0.0992	1.2054	8.9587	38.5913

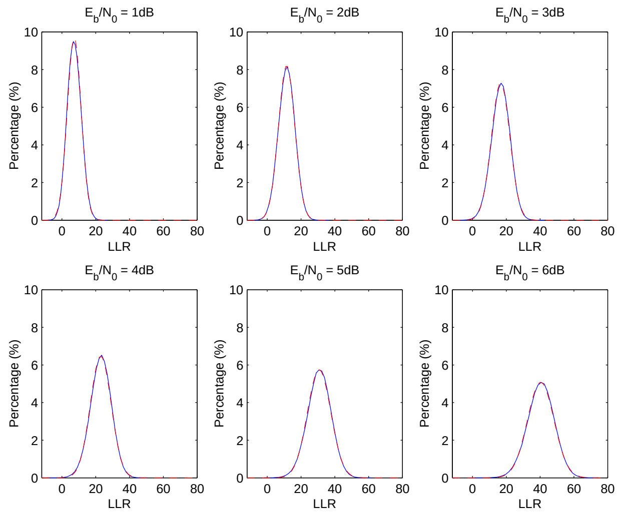
(b) *Unknown σ^2.* One difference between the BCJR algorithm and the Viterbi algorithm is that we need to know the channel variance in the BCJR algorithm. We might wonder, if our guess of σ^2 is not the same as the real value of σ^2, how well the BCJR algorithm would perform. I tested the performance of the BCJR algorithm when our guess for σ^2 is $10^{0.1}$ larger than the real value, that is, our guess for E_b/N_0 is 1dB below the true value. Figure 3 shows that (compared to Figure 2) the distribution of LLR of $E_b/N_0 = \lambda$dB seems to have the same mean and smaller variance as the distribution of LLR of $E_b/N_0 = (\lambda - 1)$dB in Figure 2(a). However, the BER and BER_t changes little. (See table below.)

E_b/N_0 (dB)	6	5	4	3	2	1
Actual error	0.0014	0.0248	0.2400	1.8032	9.4012	34.3674
Gaussian error	0.000142	0.00511	0.0994	1.2449	9.4592	40.6020

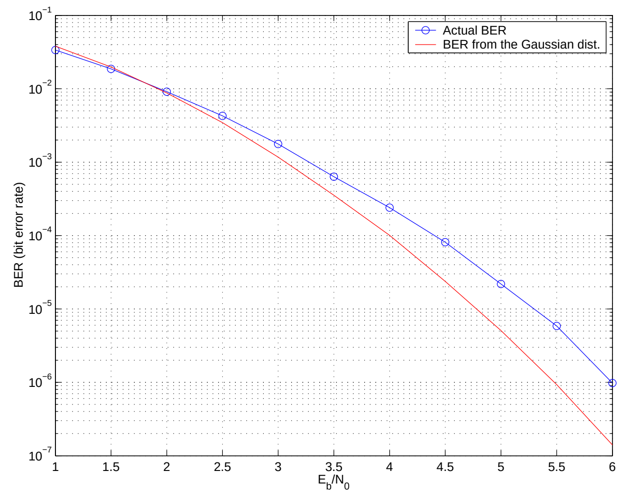
(c) *Near-linear relationship.* Notice that the mean and the variance of the adjusted LLR both increase with E_b/N_0. It is interesting to find (Figure 4) a simple linear relationship between the mean and the variance of the adjusted LLR. However, a common feature of the points in Figure 4 is that points with smaller mean and larger mean have larger slope than points between them. My guess of the real variation vs. mean curve is that it is a little like tan — with bounds in both directions for the mean. We know that the mean of the adjusted LLR should not be negative. I also guess that the upper bound for the mean is related to the free distance of the code.

(d) *Other random number generators.* I also tried some other random number generators, including the one with period 32768. The resulting BER's are similar to those got with `ran1`, though differences do exist. However, I did not test all those RNGs throughly.

(e) *Asymmetry of LLR.* We assumed that LLR satisfies the Gaussian distribution. However, after careful observation of Figure 1(a) (of course, we need a larger plot. see Figure 5), the LLR distribution is not symmetric. The left half-part is wider when E_b/N_0 is big, thus the BER calculated by (7) is smaller than the real one; when E_b/N_0 is small, the right half-part is wider thus the BER by the Gaussian is larger than the real one.More precisely, when E_b/N_0 is around 1.8, the LLR distribution is roughly symmetric. Now,

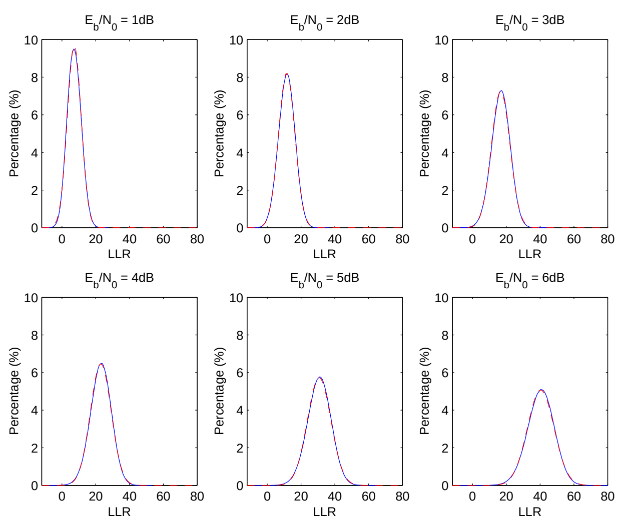


(a) Distributions of LLR

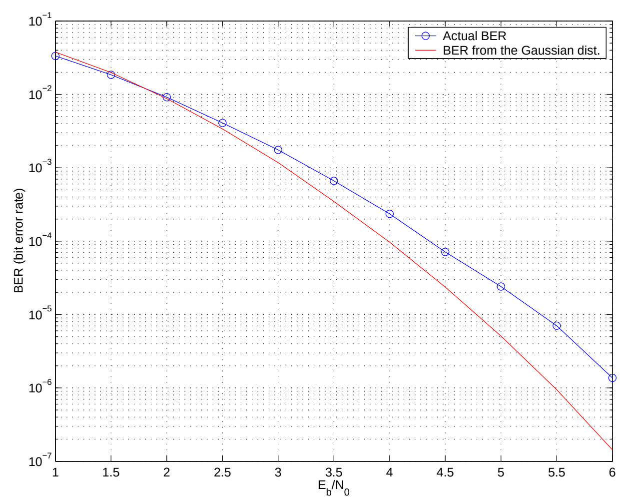


(b) BER vs. E_b/N_0

Figure 1: Result plots from 5000 runs of the BCJR decoding algorithm. `ran1()` in [Press et al., 1992] is used. (a) In each subplot, the blue curve outlines the distribution (histogram with 100 bins) of adjusted LLR under corresponding E_b/N_0, and the red curve is the Gaussian distribution with mean and variance from the real LLR data. (b) The blue circles give the average BER for 11 different tests. The red curve is the BER calculated by (7).

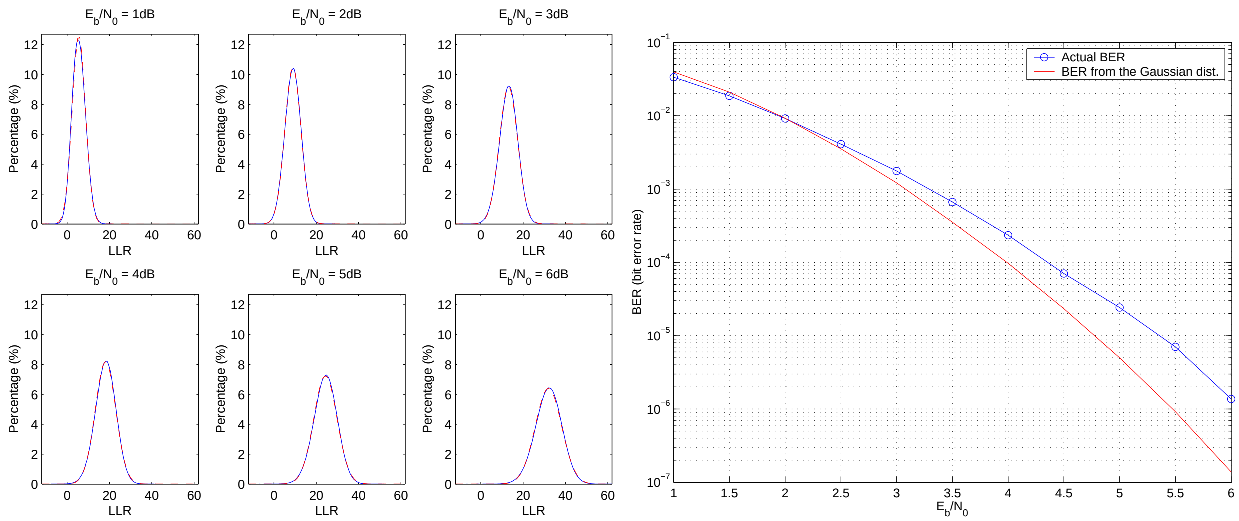


(a) Distributions of LLR



(b) BER vs. E_b/N_0

Figure 2: Result plots when the codeword is randomly generated. See Figure 1 for details.



(a) Distributions of LLR (b) BER vs. E_b/N_0

Figure 3: Result plots when the codeword is randomly generated and the guess of E_b/N_0 is 1dB lower that the true value. See Figure 1 for details.

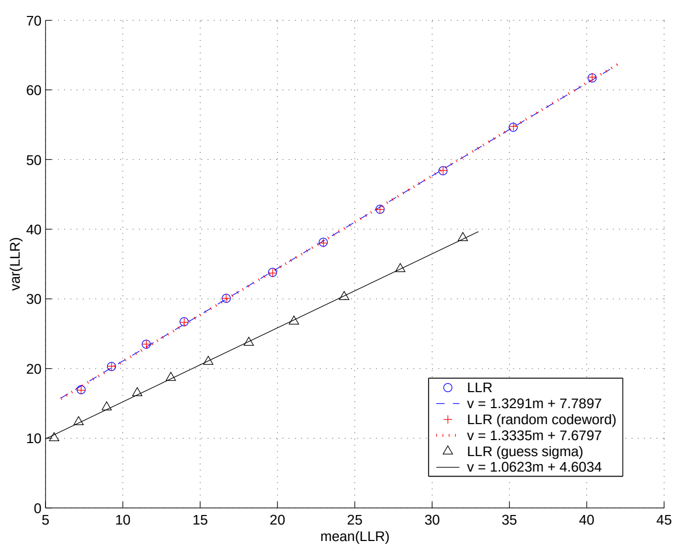


Figure 4: Near-linear relationship between the mean and the variance of the adjusted LLR. Note that the lines are linear regressions of corresponding points.

from Figure 1(b) we can see those two BERs coincide at $E_b/N_0 \approx 1.8$. What's more, in Figure 4, the variance is now twice of the mean.

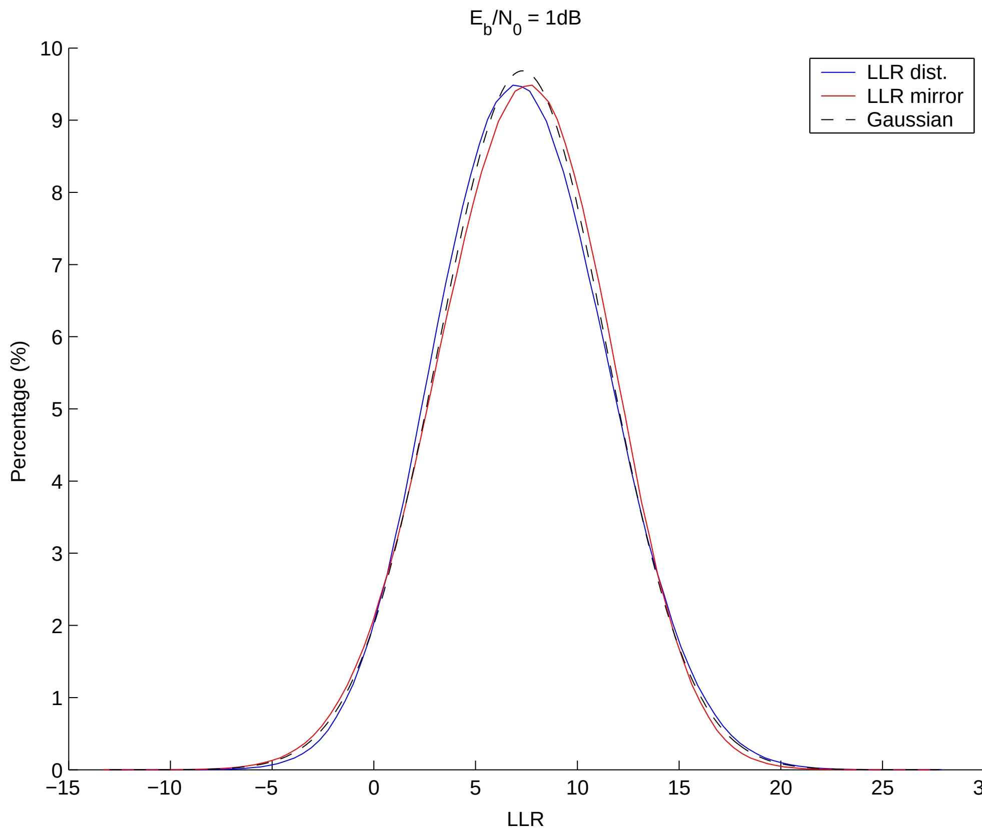


Figure 5: The adjusted LLR distribution is asymmetric ($E_b/N_0 = 1$dB, other conditions are same as Figure 1(a)) . The blue curve is the actual distribution of LLR, and the red curve is the reflection of the blue one about the mean. The black dashed curve is the Gaussian distribution with the same mean and variance.

References

[Berrou et al., 1993] Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes (1). In *Proceedings of IEEE International Communications Conference (ICC'93)*, volume 2, pages 1064–1070, Geneva, Switzerland.

[McEliece, 2001] McEliece, R. J. (2001). The forward-backward algorithm. Handout for the course EE/Ma 127c Error-Correcting Codes.

[McEliece et al., 1998] McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998). Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition.