EE/Ma 127c Error-Correcting Codes - Project 2 Turbo Code

Ling Li, ling@cs.caltech.edu

June 1, 2001

Two similar Turbo codes, the rate $\frac{1}{3}$ code in the project and the rate $\frac{1}{2}$ code in [1], were studied. Two soft-in-soft-out (SISO) decoding methods were tried as the block of the iterative decoding algorithm. Figure 1 shows the structure of the encoder and the decoder. After several iterations,

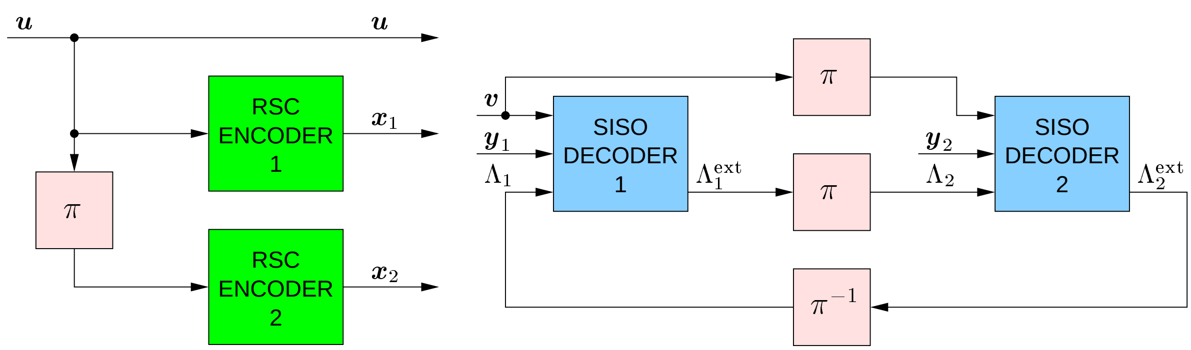



Figure 1: The structures of the encoder and the iterative decoder. $\boldsymbol{v}$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ are the noisy versions of $\boldsymbol{u}$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$. $\Lambda^{\text{ext}}$ is the extrinsic information and $\Lambda$ is the *a priori* probability.

the decoder computes $LLR_2 = \Lambda_2 + \Lambda_2^{\text{ext}}$ and outputs the decision.

The two SISO decoding methods are the BCJR [1, 3] and the message passing [4]. The difference between these two methods is in the calculation of $LLR_i$, or $\Lambda_i^{\text{ext}}$, since $\Lambda_i^{\text{ext}}$. In BCJR algorithm,

$$LLR_i(t) = \log \frac{\Pr\{u(t) = 0 | \boldsymbol{v}, \boldsymbol{y}_i\}}{\Pr\{u(t) = 1 | \boldsymbol{v}, \boldsymbol{y}_i\}},$$

while in the message passing,

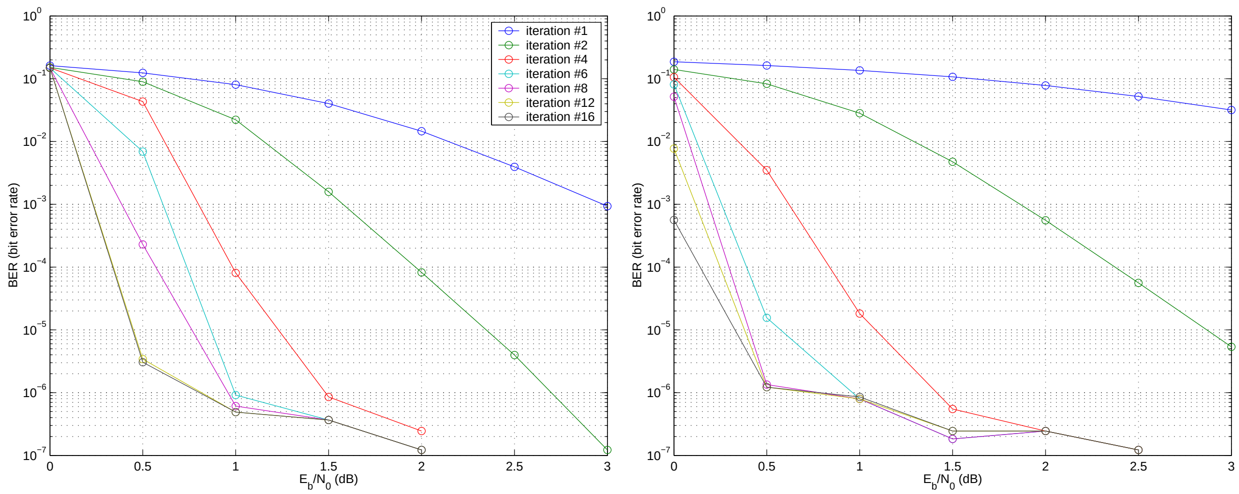
$$LLR_i(t) = \log \frac{\Pr\{u(t) = 0 | \boldsymbol{y}_i\}}{\Pr\{u(t) = 1 | \boldsymbol{y}_i\}} + \log \frac{\Pr\{\boldsymbol{v} | u(t) = 0\}}{\Pr\{\boldsymbol{v} | u(t) = 1\}}. \tag{1}$$

Details will be discussed later.

The pseudo-random interlever $\pi$ is generated by the linear congruential below:

$$\pi(i) = b \cdot a^i \pmod{p}, \quad i = 1, \ldots, p-1.$$

In this project, $b = 529$, $a = 25643$, and $p = 65497$. Thus the information word is naturally of length $k = p - 1$.




(a) Using BCJR

(b) Using message passing

Figure 2: $BER$ vs. $E_b/N_0$ for the code in the project ($r = 1/3$). Some combinations of iteration number and $E_b/N_0$ have $BER = 0$, thus could not be shown in the plots.

The bit error ratios (BERs) under different $E_b/N_0$ and iteration numbers are shown in Figure 2. For each value of $E_b/N_0 = 0, 0.5, \ldots, 3.0$dB, 250 tests with independent noise were performed. Then the average BER over those 250 tests and $k$ information bits at different iterations are calculated. From the plots, we can see that:

- Generally, with the same energy, more iterations leads to less error. Though there are counter examples, such as when $E_b/N_0 = 1.5$dB and message passing is used, the $BER$ of iteration #16 is larger than that of iteration #8 (the noisy codewords are the same), I believe that this is just because of the float round-off.
- For each value of $E_b/N_0$, there seems to be a lower limit for $BER$. That is, no matter how many iterations of decoding are performed, there is a limit $b(E_b/N_0)$ that the $BER$ can be arbitrarily close to but not lower than.

These properties can be seen more clearly from Figure 3 on page 3 and Figure 4 on page 4, which show the probability density functions of $LLR_2$. For large $E_b/N_0$, the $LLR_2$ curve with iteration #16 seems to be the limit of other curves.[1] The probability density functions of the extrinsic information $\Lambda_2^{\text{ext}}$ are also included, in Figure 5 on page 5 and Figure 6 on page 6. Note that those plots are similar to Fig. 7 in [3], but totally different from Fig. 6 in [1].

- For small number of iterations, the BCJR algorithm seems to perform better than the message passing method, since the $BER$ curves in Figure 2(a) decrease much faster than those in Figure 2(b), when the number of iterations is small.

[1]This point may not be true. See [2].

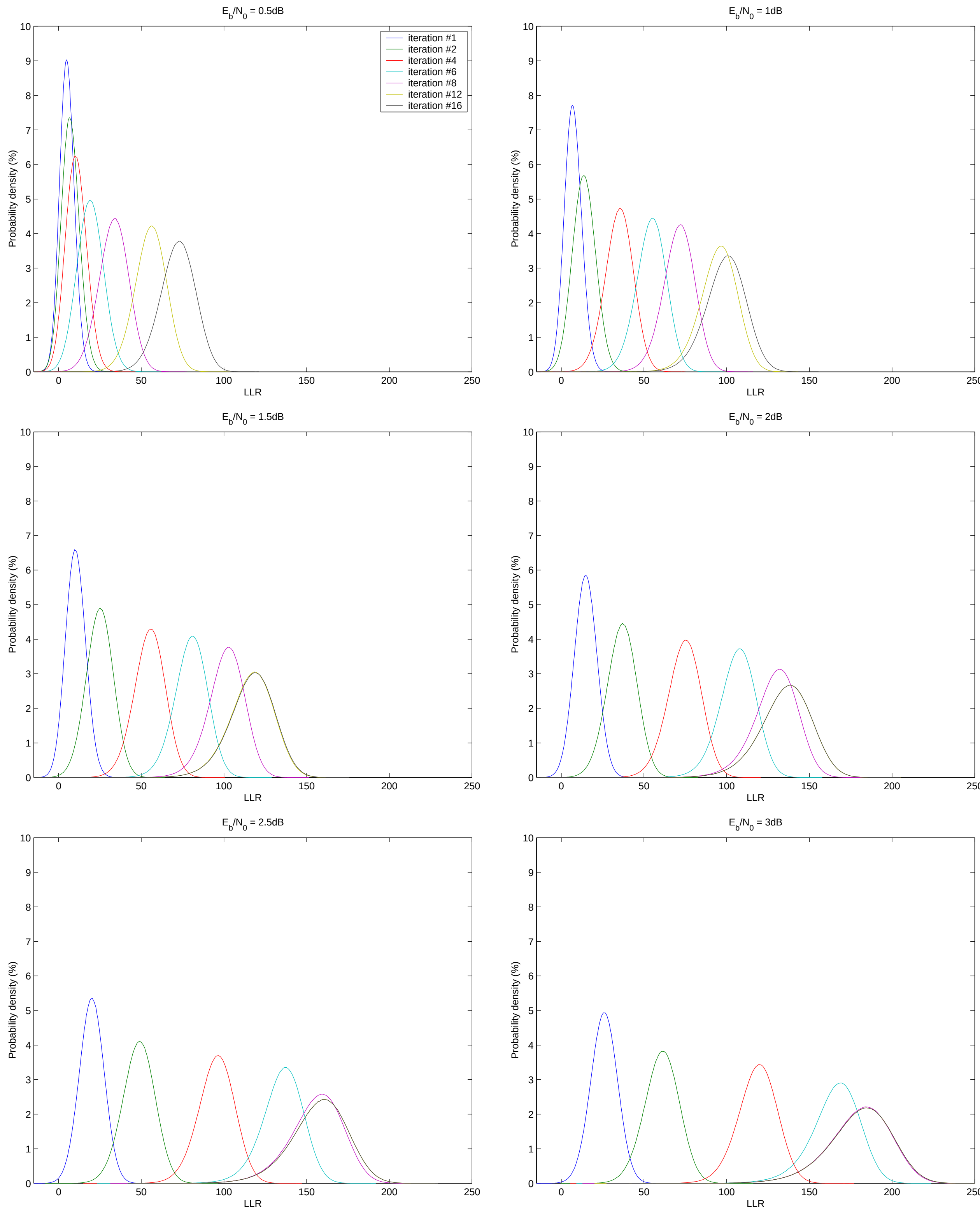



Figure 3: $LLR_2$ distribution for the code in project. BCJR method was used. $n = 65496$, $r = 1/3$, average of 250 tests.

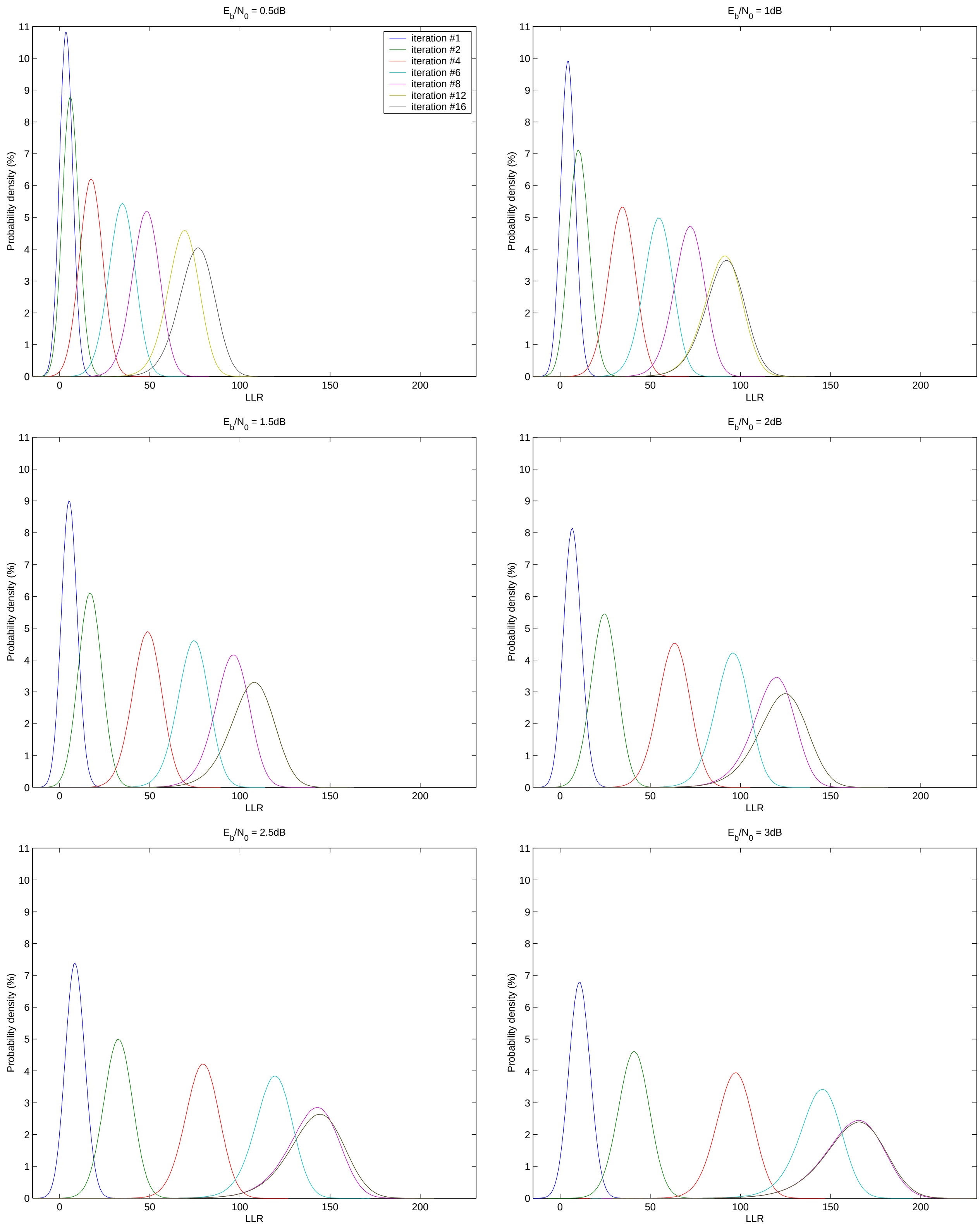



Figure 4: $LLR_2$ distribution for the code in project. Message passing method was used. $n = 65496$, $r = 1/3$, average of 250 tests.

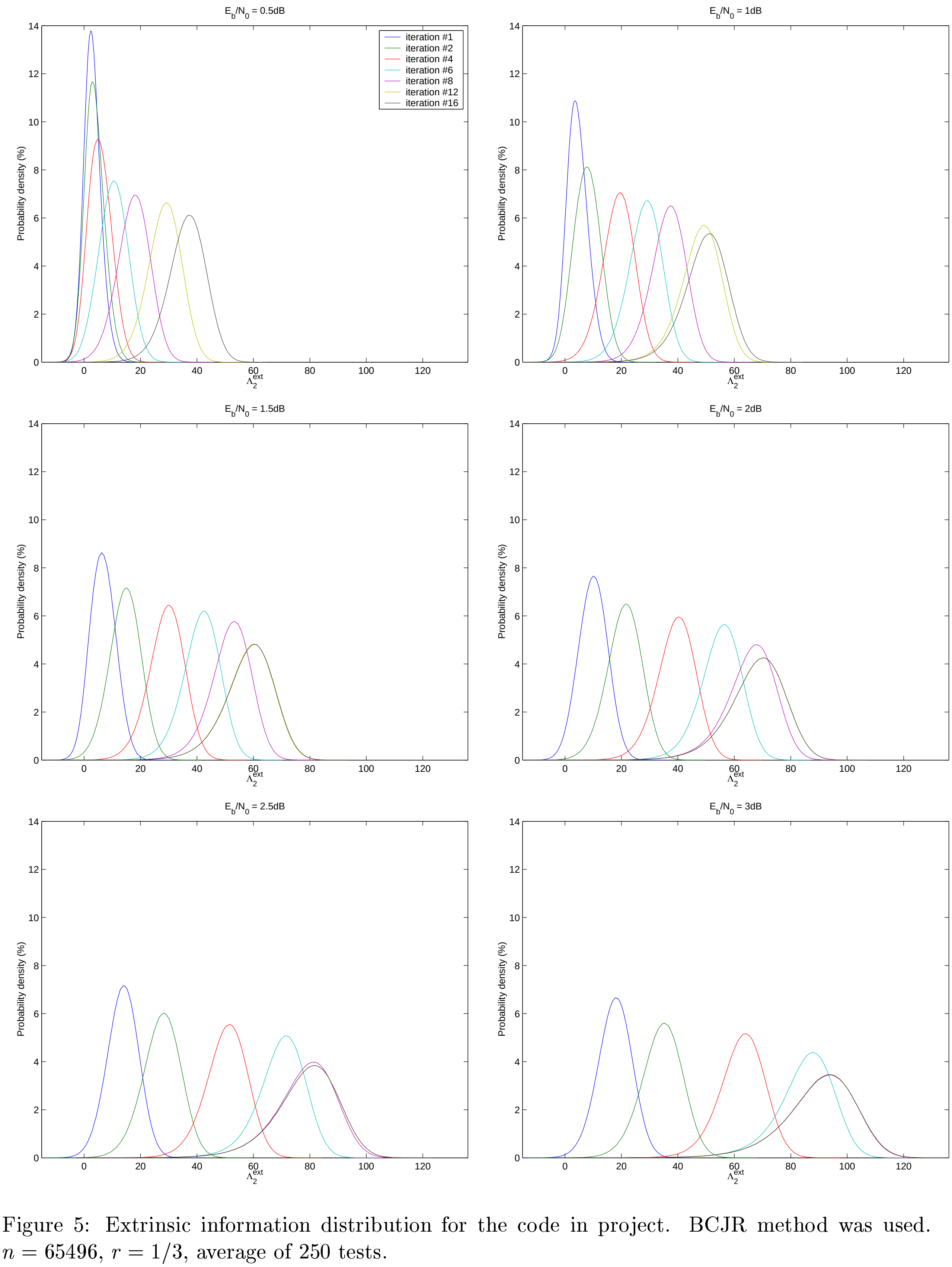



Figure 5: Extrinsic information distribution for the code in project. BCJR method was used. $n = 65496$, $r = 1/3$, average of 250 tests.

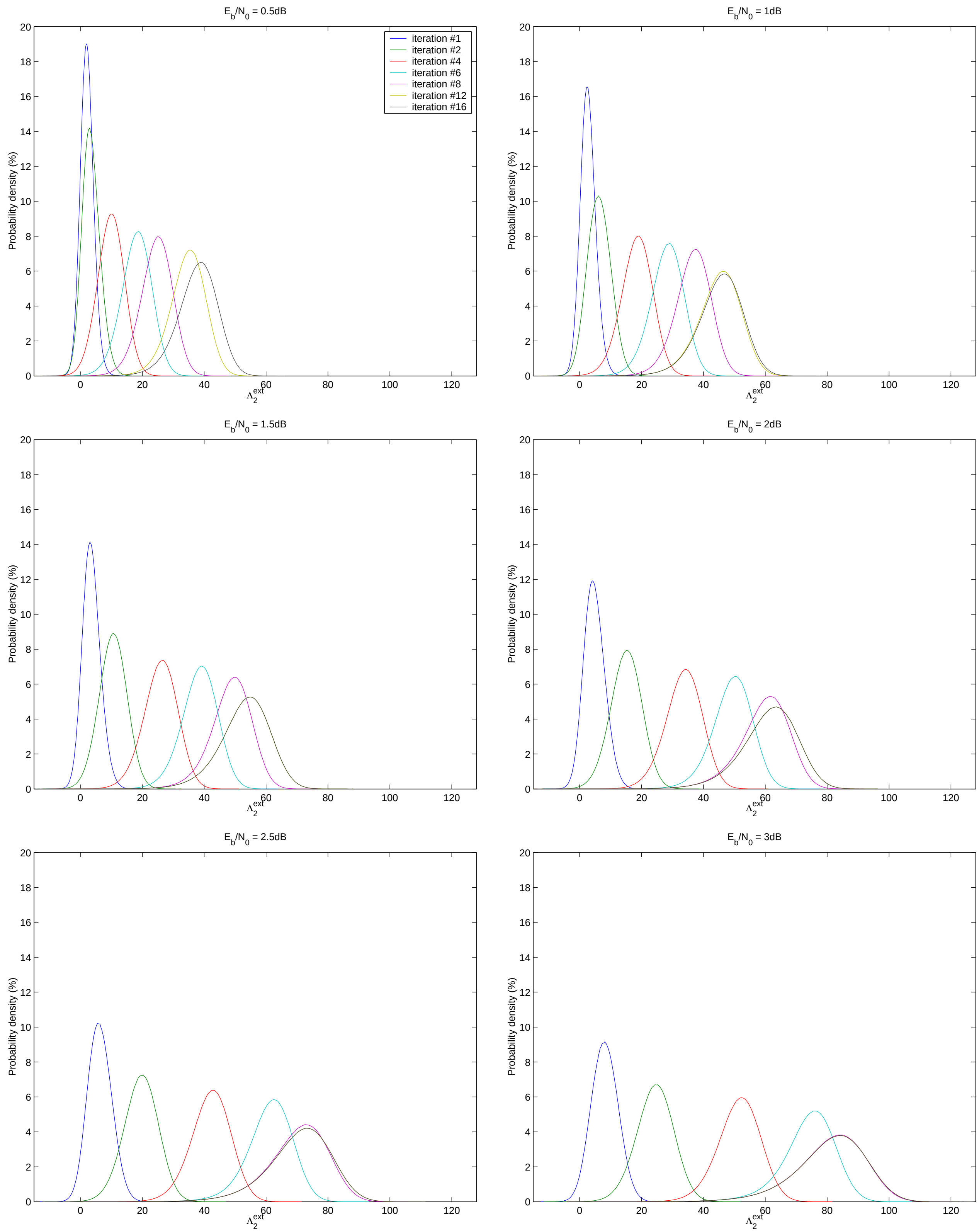



Figure 6: Extrinsic information distribution for the code in project. Message passing method was used. $n = 65496$, $r = 1/3$, average of 250 tests.

- However, the $BER$ limit of the message passing at low $E_b/N_0$ is much lower than that of the BCJR algorithm.

The reason for above observations, I guess, may relate to the magnitude of the extrinsic information. Averagely, the extrinsic information in message passing is smaller than that in BCJR algorithm, which can be seen from Figure 5 and 6. When the channel is too noisy ($E_b/N_0$ is small), a stronger extrinsic information may prevent the decoder to recover from errors of previous iterations. [1] also mentioned that

> 'For low signal to noise ratios, we have sometimes noticed that BER could increase during the iterative decoding process. In order to overcome this effect, we have divide the extrinsic information $z_k$ by $\left[1+\theta\left|\tilde{\Lambda}_1(d_k)\right|\right]$ with $\theta = 0.15$.'

Similar results with the code (rate $r = \frac{1}{2}$) in [1] are shown in Figure 8–12. With same energy for a single symbol, the $E_b/N_0$ is larger for this rate $\frac{1}{2}$ code. However, the performance is worse than the rate $\frac{1}{3}$ code in the project. No wonder of this result since more information is transmitted with the latter code.

As in the 'BCJR algorithm' project, approximation of the log-sum operation (also called as boxed-plus) is used in order to speed up the calculation. The performance of the turbo decoding is sensitive to the approximation used.

- I tried using $\max(\cdot)$ as the approximation. It does well when $E_b/N_0$ is above 1dB. However, it can not decrease the number of errors when $E_b/N_0$ is 0.5dB. After adopting another better approximation, the number of errors (when $E_b/N_0 = 0.5$dB) can be decreased dramatically. The approximation I used is a 2-bit approximation.
- My program was compiled by `gcc` under Linux. With optimization (`gcc -O3`), the speed of the algorithm can be 2–3 times faster. However, the results (compared to unoptimized one) have small differences with high iteration numbers. The reason may relate to the floating point operation which may be optimized by the compiler. However, this also implies that the density evolution is sensitive to the approximation.

In [5], the authors also mentioned that log-sum approximation has big impact on the performance.

Here are some details about the encoding and decoding:

1. For the BCJR algorithm,

$$\begin{aligned}
\Pr\{u(t)=\epsilon|\boldsymbol{v},\boldsymbol{y}_i\} &= \alpha\Pr\{u(t)=\epsilon,\boldsymbol{v},\boldsymbol{y}_i\} \\
&= \alpha\sum_{\boldsymbol{u}:u(t)=\epsilon}\Pr\{\boldsymbol{u},\boldsymbol{v},\boldsymbol{y}_i\} \\
&= \alpha\sum_{\boldsymbol{u}:u(t)=\epsilon}\Pr\{\boldsymbol{v},\boldsymbol{y}_i|\boldsymbol{u}\}\Pr\{\boldsymbol{u}\}.
\end{aligned}$$

   From the first project, the branch weight (in log-likelihood ratio) can be set as $\frac{2v+x_iy_i}{\sigma^2}+\Lambda_i$ for the branch with $u=+1$, and $\frac{x_iy_i}{\sigma^2}$ for $u=-1$, where $\sigma^2 = \left(2r\frac{E_b}{N_0}\right)^{-1}$ is the variance of the channel noise and $x_i$ is assumed to be the corresponding codebit for $u$.

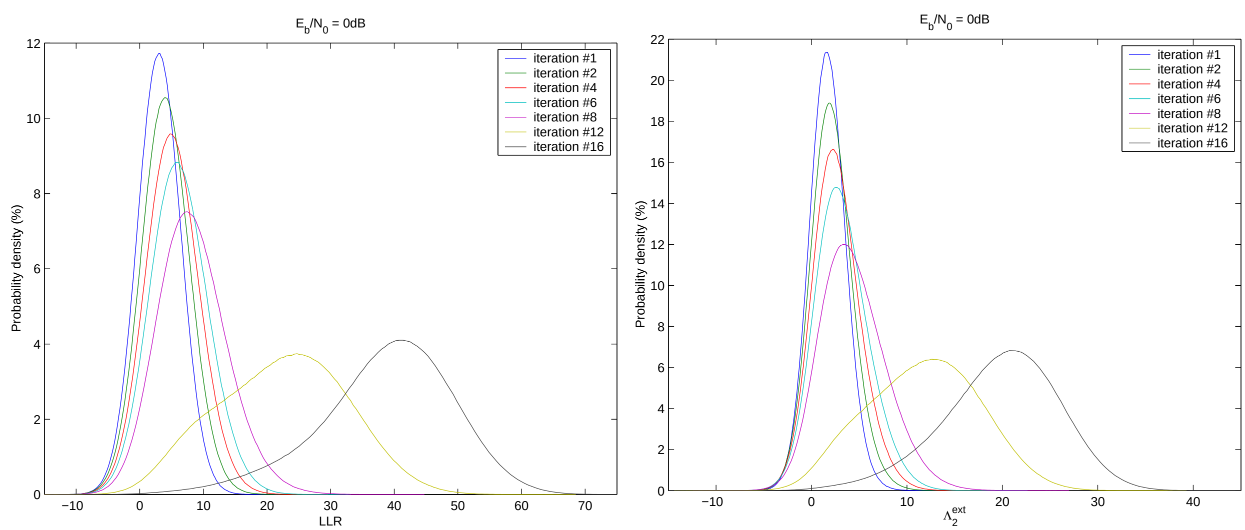
2. For message passing, the message from $\{\boldsymbol{u}, \boldsymbol{x}_i\}$ to $u(t)$ is

$$\Pr\{u(t) = \epsilon | \boldsymbol{y}_i\} = \alpha \sum_{\boldsymbol{u}:u(t)=\epsilon} \Pr\{\boldsymbol{y}_i | \boldsymbol{u}\} \Pr\{\boldsymbol{u}\}, \tag{2}$$

   and the message from $u(t)$ to $\{\boldsymbol{u}, \boldsymbol{x}_i\}$ is $\Pr\{v(t)|u(t) = \epsilon\}$, which is also $\Pr\{\boldsymbol{v}|u(t) = \epsilon\}$, since the information bits are independent. (2) can be calculated by BCJR algorithm, with branch weight (in LLR) $\frac{x_i y_i}{\sigma^2} + \Lambda_i$ and $\frac{x_i y_i}{\sigma^2}$. Then the result (also in LLR) is added by $\frac{2v(t)}{\sigma^2}$ to get (1).

3. At the end of the encoding, each encoder needs 4 dummy bits to drive its state back to 0. However, those 4 bits for encoder 1 are (usually) not the same as those for the encoder 2. We can transmit the 4 dummy bits for encoder 1 as $u(k+1)\ldots u(k+4)$. Thus $v(k+1)\ldots v(k+4)$ can be used in decoder 1. However, for decoder 2, the noisy versions of those 4 dummy bits are set to 0 since they are unknown.

An interesting thing is that there is a 'bump' in the PDF curve of $LLR_2$ and $\Lambda_2^{\text{ext}}$ when $E_b/N_0 = 0$dB




(a) $LLR_2$

(b) Extrinsic information $\Lambda_2^{\text{ext}}$

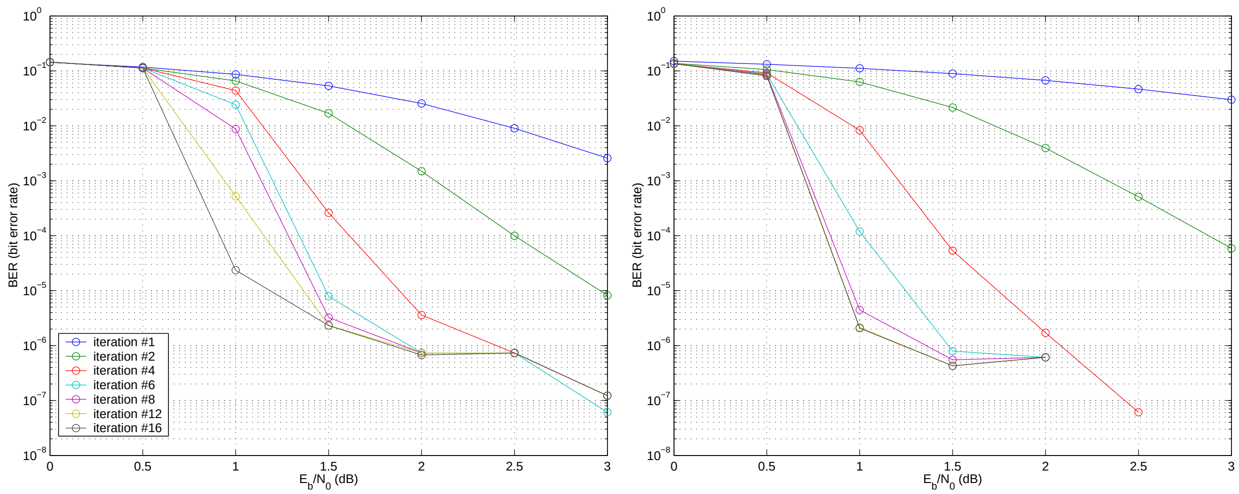
Figure 7: A bump in the curve of iteration #12, when $E_b/N_0 = 0$dB. The code is the $r = \frac{1}{3}$ code in the project and message passing is used.

and message passing is used (Figure 7).

# References


[1] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. 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, May 1993.

[2] D. Divsalar, S. Dolinar, and F. Pollara. Iterative turbo decoder analysis based on density evolution. TMO Progress Report 42-144, Jet Propulsion Laboratory, February 15, 2001.




(a) Using BCJR

(b) Using message passing

Figure 8: BER vs. $E_b/N_0$ for the $r = \frac{1}{2}$ code in [1].


[3] D. Divsalar and F. Pollara. Turbo codes for deep-space communications. TDA Progress Report 42-120, Jet Propulsion Laboratory, February 15, 1995.

[4] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, February 1998.

[5] Toshiyuki Miyauchi, Kouhei Yamamoto, and Takashi Yokokawa. High-performance programmable SISO decoder VLSI implementation for decoding turbo codes. 2001.

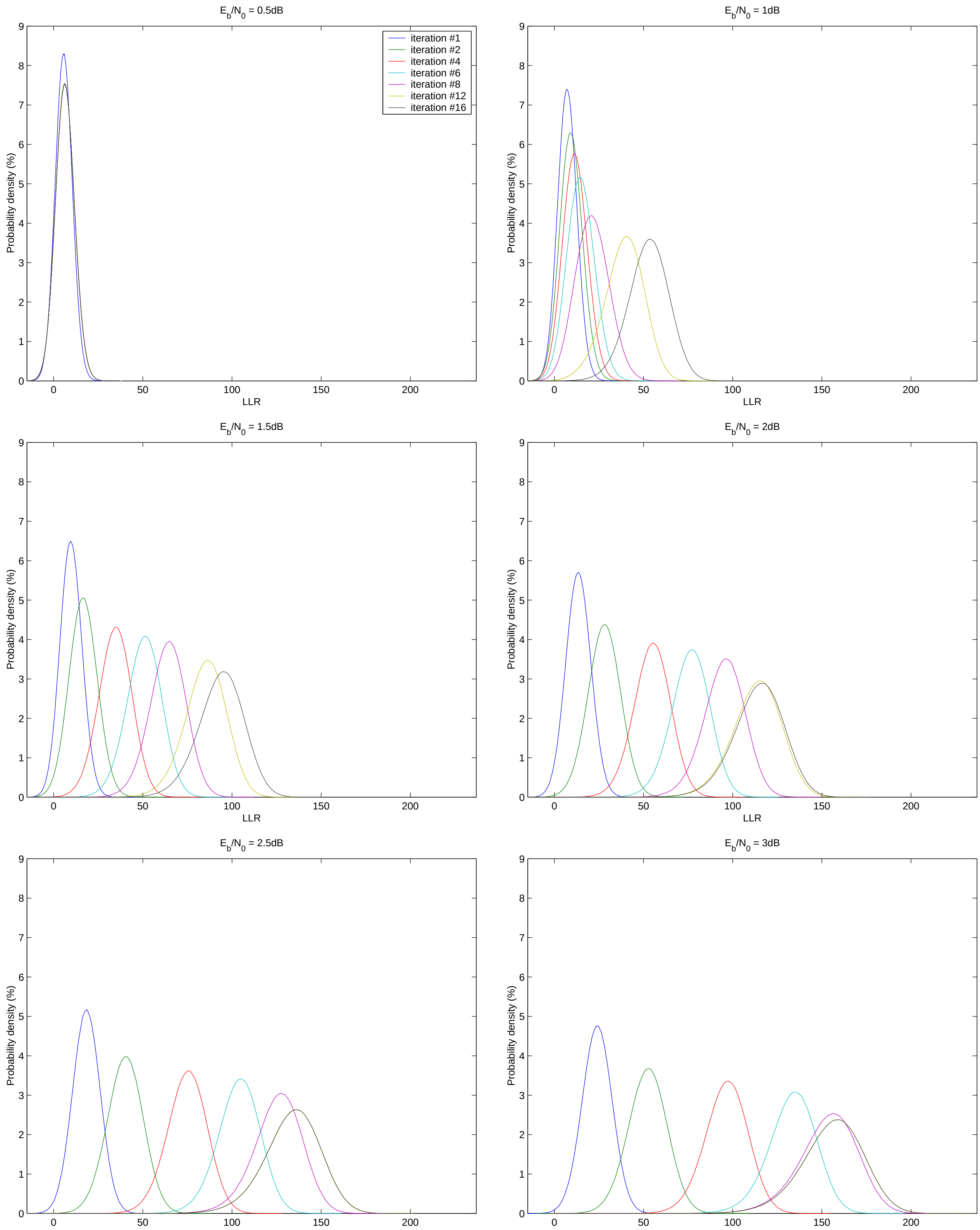



Figure 9: $LLR_2$ distribution for the code in [1]. BCJR method was used.

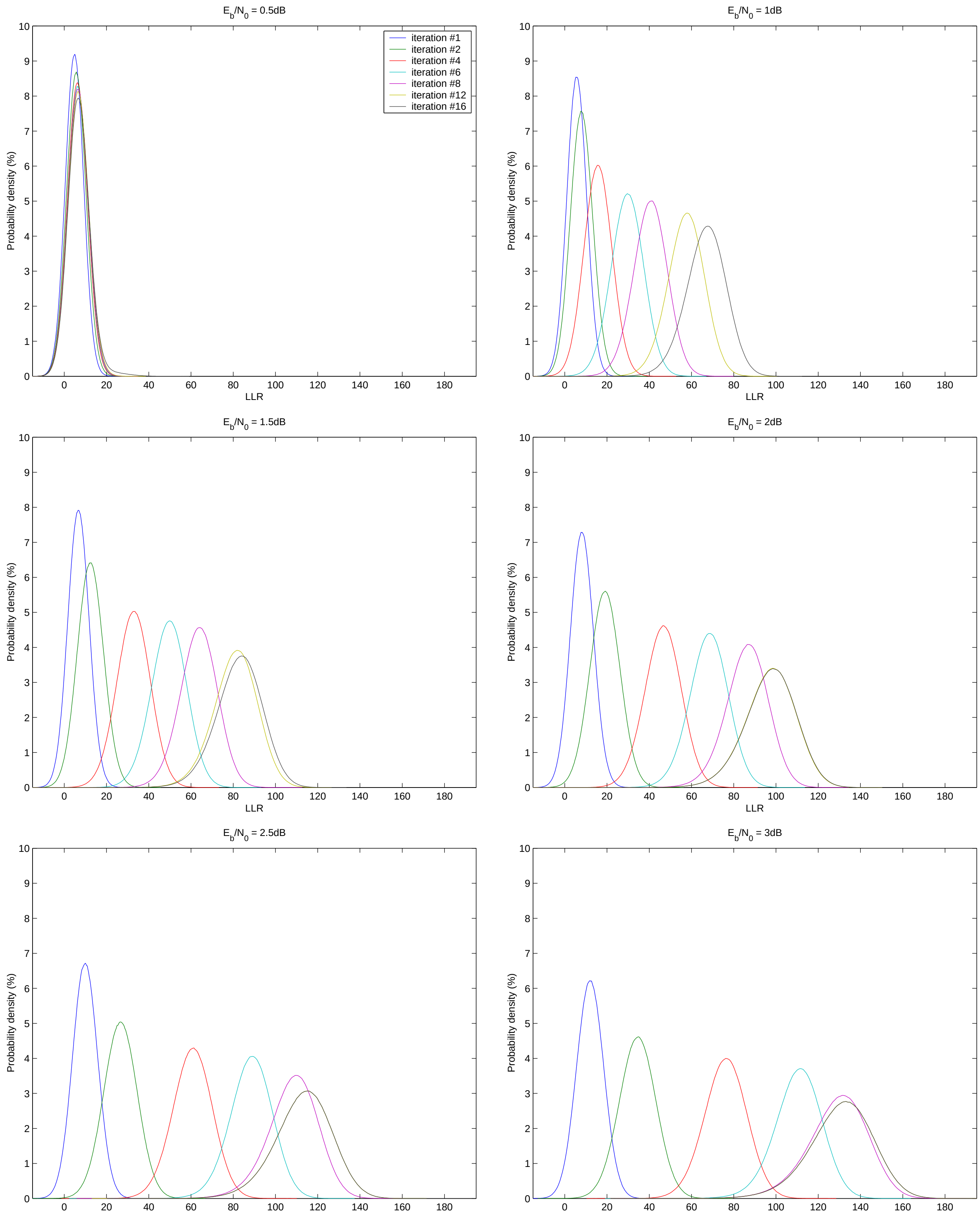



Figure 10: $LLR_2$ distribution for the code in [1]. Message passing method was used.

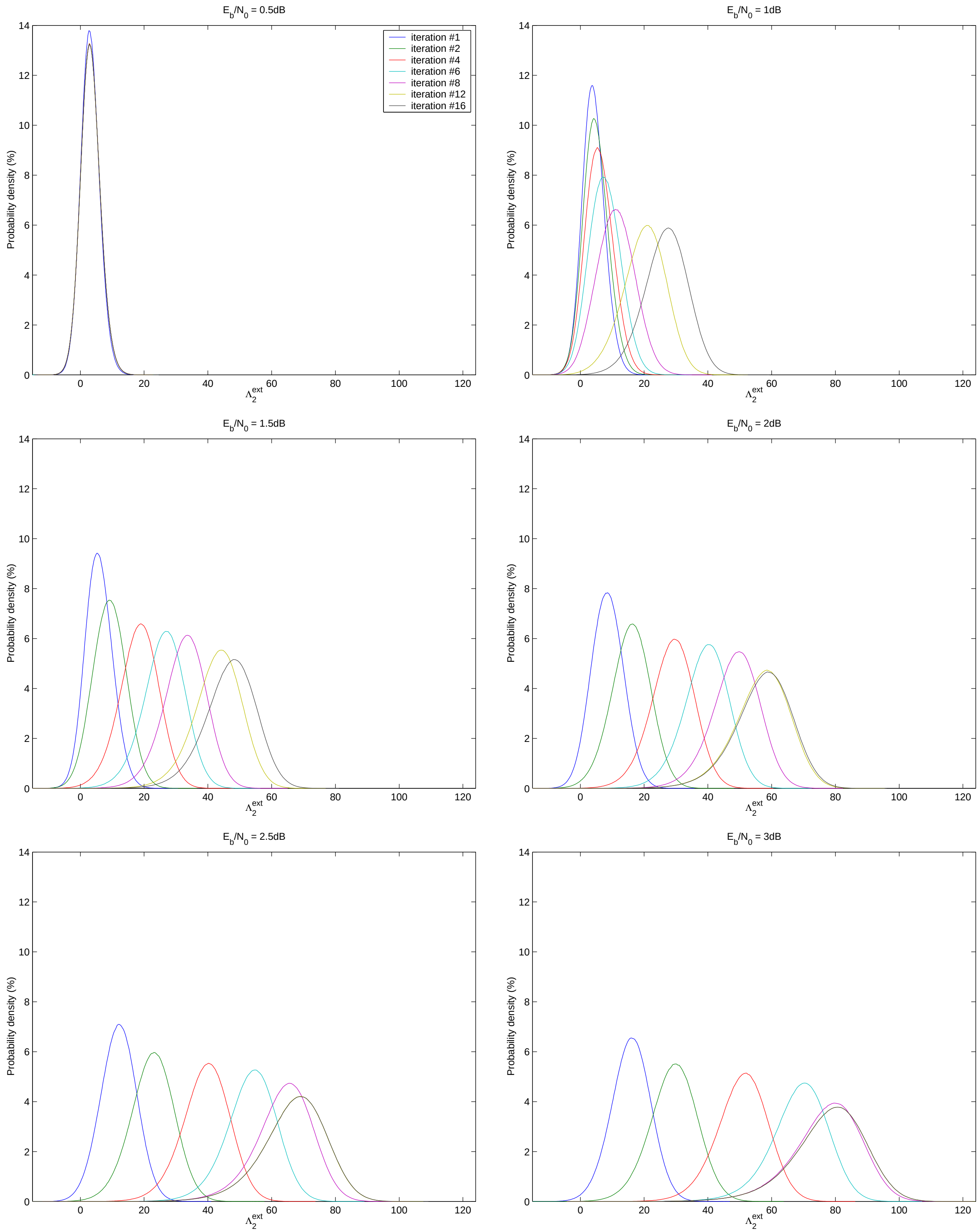



Figure 11: Extrinsic information distribution for the code in [1]. BCJR method was used.

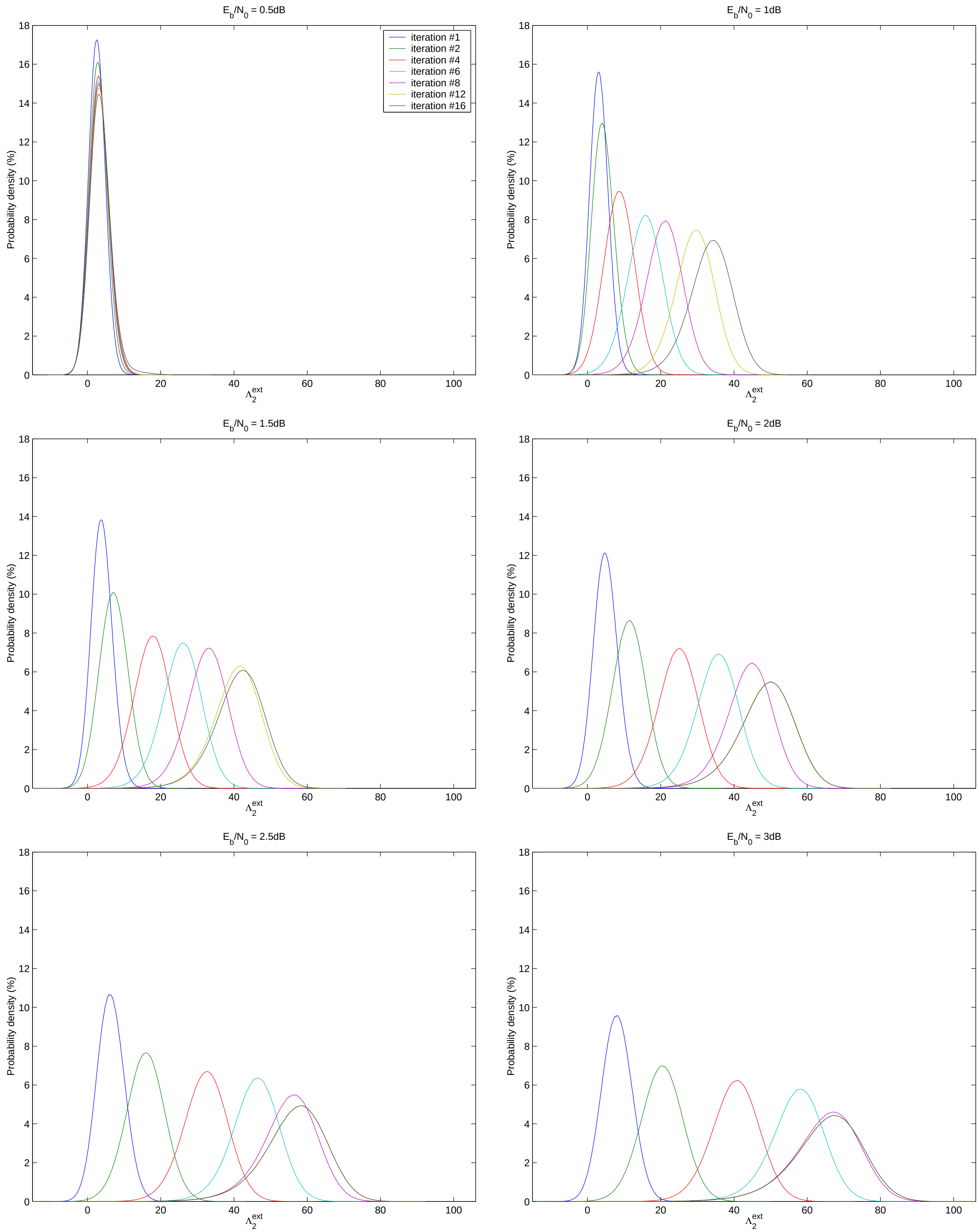



Figure 12: Extrinsic information distribution for the code in [1]. Message passing method was used.