Tutorial on Turbo Code Algorithms

Yoann Roth

2017

This work was realized in the context of my PhD, supervised by

Jean-Baptiste Doré
Laurent Ros
Vincent Berg

Contents

Edited on September 5, 2017

Author's Note

This document intends to give some precisions of the computations for two main algorithms often used when considering Turbo Codes (TCs): the Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm and the EXtrinsic Information Transfer (EXIT) chart. After a rapid introduction to the TC principle, both algorithms are intensively detailed.

The other part of this document describes the TurboCode class which can be found attached to the document. This class illustrates the principle detailed previously and allow for the creation of comprehensive scripts concerning TC.

For any suggestions, feel free to contact me at my address: yoannsroth@gmail.com.

CHAPTER 1

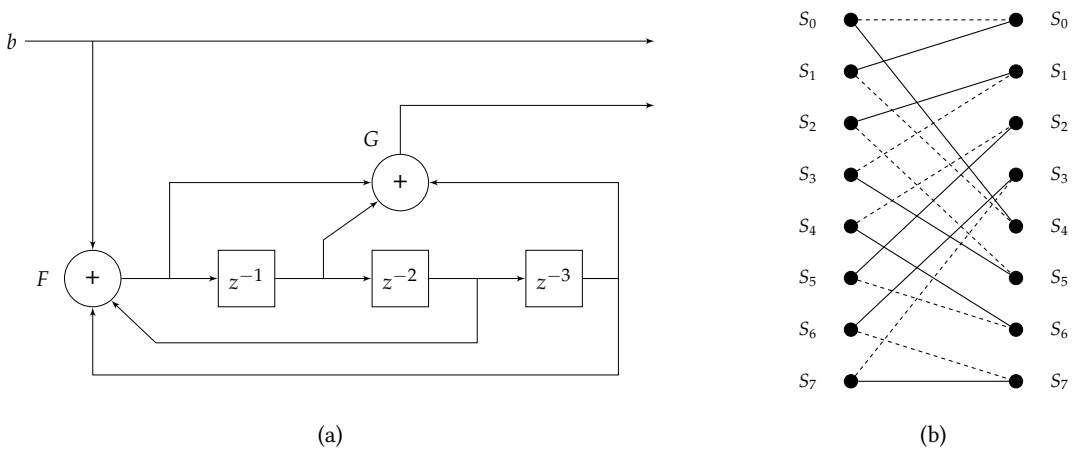
Introduction to Turbo Codes

The turbo principle was invented by Berrou, Glavieux and Thitimajshima in 1993 [1]. Their technique was rapidly shown to be able to closely approach the channel capacity, and is considered a major breakthrough in channel coding. The concept relies on the parallel concatenation of two Recursive Systematic Convolutional (RSC) encoders separated by an interleaver. The original proposition, considered here, consisted in two rate $1/2$ RSC codes. If the principle of code concatenation, proposed in [2], appeared long before the invention of TC, the novelty of the principle was mainly in the decoding procedure, with the use of iterative decoding.

This short chapter is dedicated to the introduction of the principle of turbo coding. After the rapid description of the encoding process, the principle of turbo decoding is reviewed. Typical Bit Error Rate (BER) performance are then presented for the $[13\,15]$ RSC constituent.

1.1 Encoder

A turbo encoder usually consists in the concatenation of two RSC encoder. An example of RSC encoder, the $[13\,15]$ RSC, is depicted in Figure 1.1. An alternative representation to the encoder is also its trellis representation, depicted in Figure 1.1 for this RSC. The typical parallel concatenation of a turbo encoder is depicted in Figure 1.2 (a). With the use of the interleaver, both encoders operate on the same set of information bits, but have different input sequences, and thus different output sequences. As the two encoders are systematic with rate $1/2$, the input bits are only transmitted once and the data rate is $1/3$. The encoding process is often refered to as Parallel Concatenated Convolutional Code (PCCC).



(a) (b)

Figure 1.1 – *(a) Encoder of the* $[13\,15]$ *convolutional code in its recursive form. With* b *the input bits, the output of the first adder* F *is given by the vector product* $f = [b\ z^{-1}\ z^{-2}\ z^{-3}] \otimes [1\,0\,1\,1]^\intercal$*, where* $\otimes$ *is the binary product. The binary vector* $[1\,0\,1\,1]^\intercal$ *can be represented in octal by* 13*, and is the feedback generator. The output of the adder* G *is given by* $g = [f\ z^{-1}\ z^{-2}\ z^{-3}] \otimes [1\,1\,0\,1]^\intercal$*, where the binary vector* $[1\,1\,0\,1]^\intercal$ *is* 15 *in octal: this is the feedforward generator. (b) Trellis of the* $[13\,15]$ *convolutional code in its recursive form.*

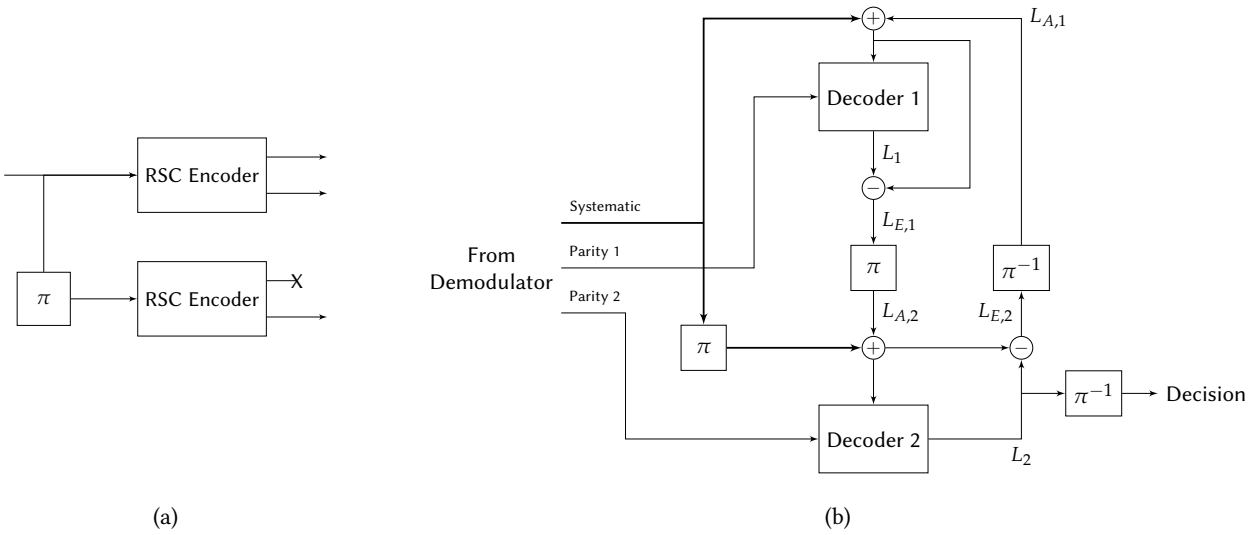


FIGURE 1.2 – *Principle of Turbo Code encoder (a) and decoder (b)*

1.2 DECODER

The architecture of the decoder is depicted in Figure 1.2 (b). It is composed of two decoders, one for each RSC code, and several interleavers. Each decoder uses the BCJR to estimate the *A Posteriori* Probability (APP) of the information bits, or more precisely the log-APP ratios of the information bits, defined as

$$L\left(b_{n,t} \,\middle|\, \boldsymbol{R}_1^N\right) = \log \frac{\Pr\left(b_{n,t} = +1 \,\middle|\, \boldsymbol{R}_1^N\right)}{\Pr\left(b_{n,t} = -1 \,\middle|\, \boldsymbol{R}_1^N\right)}, \tag{1.1}$$

where $\boldsymbol{R}_1^N = [\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N]$ is the received sequence of codewords and $b_{n,t}$ the information bit at index n of the t-th information word. In turbo decoding, unlike usual convolutional codes decoding, one decoder uses both the information from the channel and the output information generated by the other decoder. The mathematical computations for the BCJR algorithm are detailed in Chapter 2. The likelihood of a codeword is now weighted by the *a priori* information of having that codeword, coming from the other decoder. Considering the noisy codeword received at instant t $\boldsymbol{r}_t$, a codeword $\boldsymbol{c}^i$ of the alphabet and its associated information word $\boldsymbol{b}^i$, with $i \in \{0, \ldots, M-1\}$, the receiver computes, as demonstrated in Section 2.2.2, Equation (2.27)

$$p\left(\boldsymbol{r}_t \,\middle|\, \boldsymbol{c}_t^i\right) \Pr\left(\boldsymbol{c}_t^i\right) = C_{\text{st}} \exp\left\{\frac{1}{2}\sum_{k\in\mathcal{S}}\left((L(b_{k,t}) + L_A(b_{k,t}))b_{k,t}^i\right) + \frac{1}{2}\sum_{k\in\mathcal{P}} L(c_{k,t})c_{k,t}^i\right\}. \tag{1.2}$$

The sum over $\mathcal{S}$ (resp. $\mathcal{P}$) represents the sum over the indexes corresponding to the systematic positions (resp. the parity positions). There are $\log_2(M)$ systematic positions, and $N_c - \log_2(M)$ parity positions. $b_{k,t}$ refers to the bit at index k of the information word associated to the received codeword at instant t, and $c_{k,t}$ to the parity bit at index k. $L(b_{k,t})$ (resp. $L(c_{k,t})$) is the Log Likelihood Ratio (LLR) of the bit $b_{k,t}$ (resp. $c_{k,t}$), and $L_A(b_{k,t})$ the *a priori* log ratio on the information bits, defined as

$$L_A(b_{k,t}) = \log \frac{\Pr\left(b_{k,t} = +1\right)}{\Pr\left(b_{k,t} = -1\right)}. \tag{1.3}$$

Finally, C_{st} is a constant independent of i, cancelled out in further computation in the BCJR algorithm. Equation (1.2) clearly shows how the *a priori* information is included to the LLR at the systematic bits positions.

When selecting the information bit with index $k = n$ at time index t, the log-APP of the information bits given the received sequence of codewords $\boldsymbol{r}_1^N$, can be factorized to, according to Equation (2.30),

$$L\left(b_{n,t} \,\middle|\, \boldsymbol{R}_1^N\right) = L\left(b_{n,t}\right) + L_A\left(b_{n,t}\right) + L_E. \tag{1.4}$$

This means that the log-APP contains three informations: the LLR of the information bit from the channel $L\left(b_{n,t}\right)$, the *a priori* LLR $L_A\left(b_{n,t}\right)$ from the other decoder and the extrinsic information generated by the algorithm L_E

(which depends on all the other bits). Only the extrinsic information is sent to the other decoder as *a priori*, as this will be new information from its point of view. The turbo name comes from this re-injection of the information between decoders, and similar to the principle used in turbo charged car engines.

The general decoding procedure can be described in several steps:

1. The *a priori* LLR are initialized to 0 (i.e. no *a priori* information).
2. Decoder 1 sums the LLR of the systematic bits from the channel and the *a priori* LLR $L_{A,1}$, and uses the LLR of the parity bits to compute (1.2). The BCJR algorithm is applied to compute the log-APP of the information bits L_1;
3. The extrinsic information of the first decoder $L_{E,1}$ is computed, interleaved and sent as the *a priori* LLR $L_{A,2}$ to the second decoder;
4. Decoder 1 sums the interleaved LLR of the systematic bits from the channel and the *a priori* LLR $L_{A,2}$, and uses the LLR of the parity bits to compute (1.2). The BCJR algorithm is applied to compute the log-APP of the information bits L_2;
5. The extrinsic information $L_{E,2}$ of the second decoder is computed and deinterleaved, and sent back to the first decoder, which will treat it as *a priori* information $L_{A,1}$.

The steps 2 to 5 correspond to one iteration. They can be repeated as many times as necessary. After a certain number of iterations, the APP of the information bits from the second decoder L_2 can be used to make a hard decision on the bits.

1.3 Typical Performance

The typical performance of a TC using Monte Carlo simulations is depicted in Figure 1.3. The RSC constituent is the [13 15] Convolutional Code (CC), which encoder and trellis are depicted in Figure 1.1. The performance metric used is the BER, defined as the ratio of erroneous bits over the total number of bits. The BER was computed for several values of E_b/N_0 and 4 values for the information block size Q (also equal to the interleaver size). The performance of uncoded Binary Phase Shift Keying (BPSK) is also depicted, demonstrating the gain offered by the TC. The BER performance of a TC is generally characterized by 3 regions:

- The *non-convergence region*, which corresponds to low values of E_b/N_0, where the BER stays high and does not evolve;
- The *waterfall region*, where the BER drops significantly. The starting E_b/N_0 value of the waterfall can be estimated by computing the threshold of the decoder (also called the turbo cliff position). It is the minimum

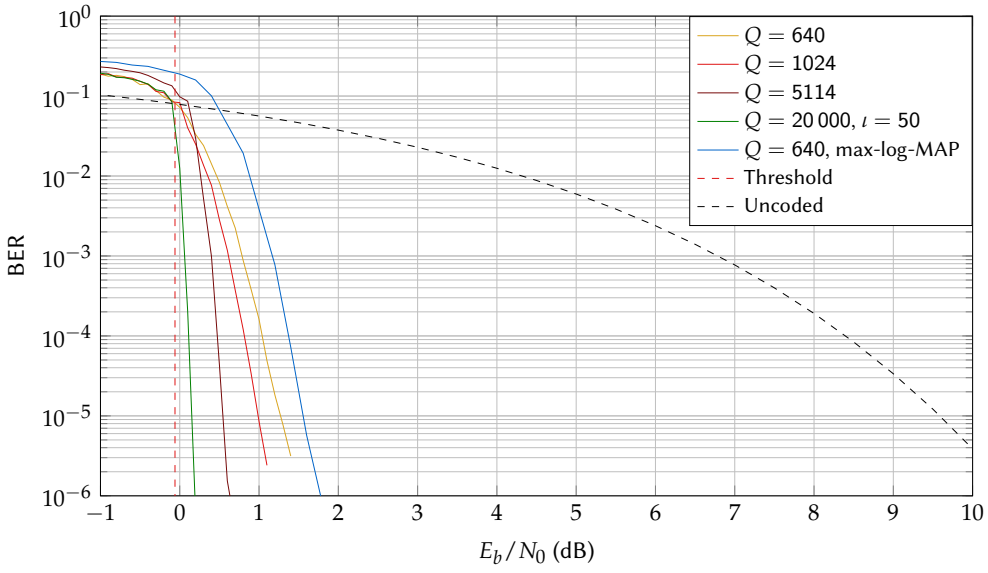


Figure 1.3 – *Performance of the* [13 15] *TC for various interleaver sizes Q, under the AWGN channel. A total of $\iota = 10$ iterations are performed unless stated otherwise.*

level of E_b/N_0 required for the decoder to be able to correct all the erroneous bits, regardless of the interleaver size and the number of iterations performed. It represents the asymptotic behavior of the decoder. Evaluating the threshold can be done using the EXIT chart tool [3], which is presented for the case of TC in Chapter 3. The estimated threshold for the code considered is given in Figure 1.3;

- The *error floor region*, which is the region of E_b/N_0 where the BER performance flattens compared to the waterfall and decreases slowly with E_b/N_0.

Both the waterfall and the error floor regions depend on the interleaver size Q and the choice of constituent RSC codes. The error floor can be upper bounded using the distance spectrum of the code [4], and the bound can be approximated using only the low-weight codewords [5]. Concerning the waterfall, the dependence with Q corresponds to the loss of diversity when shortening the information block length [3]; this diversity is introduced thanks to both versions (one interleaved, the other not) of the same message.

The EXIT chart is a very useful tool when it comes to code or system design and analysis [6], and its interpretation gives indications on the number of iterations required to correct the maximum number of bits; during the waterfall, a large number of iterations are required when long interleaver sizes are considered, as a large number of information exchanges are required for the decoder to correct erroneous bits. During the error floor, the decoder converges in only a few iterations, and the performance is dominated by the low-weight codewords.

In practical systems, the computation of the BCJR can reveal itself to be tricky because of the presence of exponential terms and multiplications. While the use of the Soft-Output Viterbi Algorithm (SOVA) algorithm to decode the trellis could be considered [7, 8], suboptimal Maximum *A Posteriori* (MAP) algorithm can be used such as the log-MAP and max-log-MAP algorithms [9]. The max-log-MAP algorithm relies on the simplification

$$\log\left(\sum_i e^{x_i}\right) \simeq \max_i(x_i), \tag{1.5}$$

and all computations can be done using only additions and max operations. In Figure 1.3, the BER performance of the decoder when using the suboptimal max-log-MAP is depicted, for the interleaver size $Q = 640$. The simplification is done at the expense of some performance loss, approximately 0.3dB for a BER of 10^{-5}, but this can be overcome with the use of weighting functions on the exchanges of extrinsic information inside the decoder [10].

CHAPTER 2

Optimal Trellis-Decoding

The BCJR was initially proposed in [11], and revisited in [1] for its use in TCs. This algorithm computes the APP of the information bits given the received codewords. This chapter is dedicated to the description of the initial BCJR algorithm. Its application for channel coding is also reviewed.

2.1 *A Posteriori* Probabilities of Transitions

We consider a discrete time Markov chain with N_s possible states and N transitions. At time t the state is S_t, with $S_t \in \{0, \ldots, N_s - 1\}$ and $t \in \{0, \ldots, N\}$. By convention, $S_0 = S_N = 0$. The N observations of the transitions are denoted $\boldsymbol{R}_1^N = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t, \ldots, \boldsymbol{r}_N\}$.

The goal of the algorithm is to determine the APP of the transitions $\Pr\left(S_{t-1} = s',\, S_t = s \,\middle|\, \boldsymbol{R}_1^N\right)$, with $\{s, s'\} \in \{0, \ldots, N_s - 1\}^2$, which can be expressed, using Bayes' law,

$$\Pr\left(S_{t-1} = s',\, S_t = s \,\middle|\, \boldsymbol{R}_1^N\right) = \frac{p\left(\boldsymbol{R}_1^N \,\middle|\, S_{t-1} = s',\, S_t = s\right) \Pr\left(S_{t-1} = s',\, S_t = s\right)}{\Pr\left(\boldsymbol{R}_1^N\right)}. \tag{2.1}$$

Since the probability $\Pr\left(\boldsymbol{R}_1^N\right)$ is constant for an observation $\boldsymbol{R}_1^N$, we will derive the numerator in order to have the expression of the APP' transition. Using total probabilities, the numerator can be expressed

$$\begin{aligned} p\left(S_{t-1} = s',\, S_t = s,\, \boldsymbol{R}_1^N\right) &= p\left(S_{t-1} = s',\, S_t = s,\, \boldsymbol{R}_1^{t-1},\, \boldsymbol{r}_t,\, \boldsymbol{R}_{t+1}^N\right) \\ &= p\left(S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) p\left(S_t = s,\, \boldsymbol{r}_t,\, \boldsymbol{R}_{t+1}^N \,\middle|\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) \\ &= p\left(S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) \\ &\quad \cdot p\left(\boldsymbol{R}_{t+1}^N \,\middle|\, S_t = s,\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1},\, \boldsymbol{r}_t\right). \end{aligned} \tag{2.2}$$

As we considered a Markov process, if the state S_t is known, events prior to time t do not depend on the observations until t. The following terms can thus be simplified as

$$\begin{aligned} p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) &= p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s'\right) \\ p\left(\boldsymbol{R}_{t+1}^N \,\middle|\, S_t = s,\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1},\, \boldsymbol{r}_t\right) &= p\left(\boldsymbol{R}_{t+1}^N \,\middle|\, S_t = s\right). \end{aligned}$$

We define the function gamma as

$$\gamma_t(s', s) = p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s'\right) \tag{2.3}$$

which is the joint Probability Density Function (PDF) to be in state s at index t and to observe $\boldsymbol{r}_t$, given that the state at index $t-1$ was s'. This PDF is related to the likelihood of the transition from state s to s'.

We define the probability function α such that

$$\alpha_t(s) = p\left(S_k = s,\, \boldsymbol{R}_1^t\right), \tag{2.4}$$

which is the joint PDF of being in state s at time t and to observe the received sequence up to index t, i.e. the "past" of the sequence.

We define function β as

$$\beta_t(s) = p\left(\boldsymbol{R}_{t+1}^{N} \,\middle|\, S_t = s\right), \tag{2.5}$$

which is the PDF of the sequence of received symbols after index t (i.e. the "future" of the sequence) given that the state at time t was s.

2.1.1 Computation of α

The value of α can be obtained recursively by summing all the possibilities for the previous states S_{t-1}

$$\begin{aligned}
\alpha_t(s) &= \sum_{s'=0}^{N_s-1} p\left(S_{t-1} = s',\, S_t = s,\, \boldsymbol{R}_1^t\right) = \sum_{s'=0}^{N_s-1} p\left(S_{t-1} = s',\, S_t = s,\, \boldsymbol{R}_1^{t-1},\, \boldsymbol{r}_t\right) \\
&= \sum_{s'=0}^{N_s-1} p\left(S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) \\
&= \sum_{s'=0}^{N_s-1} p\left(S_{t-1} = s',\, \boldsymbol{R}_1^{t-1}\right) p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s'\right) \\
&= \sum_{s'=0}^{N_s-1} \alpha_{t-1}(s')\gamma_t(s', s)
\end{aligned} \tag{2.6}$$

where the equality between the second and third line comes from the Markov property. The values of α are thus obtained by performing a forward recursion on the trellis.

2.1.2 Computation of β

The value of β is obtained by summing all the possibilities for the next states S_{t+1}

$$\begin{aligned}
\beta_t(s) &= \sum_{s'=0}^{N_s-1} p\left(S_{t+1} = s',\, \boldsymbol{R}_{t+1}^{N} \,\middle|\, S_t = s\right) = \sum_{s'=0}^{N_s-1} p\left(S_{t+1} = s',\, \boldsymbol{r}_{t+1},\, \boldsymbol{R}_{t+2}^{N} \,\middle|\, S_t = s\right) \\
&= \sum_{s'=0}^{N_s-1} p\left(S_{t+1} = s',\, \boldsymbol{r}_{t+1} \,\middle|\, S_t = s\right) p\left(\boldsymbol{R}_{t+2}^{N} \,\middle|\, S_t = s,\, S_{t+1} = s',\, \boldsymbol{r}_{t+1}\right) \\
&= \sum_{s'=0}^{N_s-1} p\left(S_{t+1} = s',\, \boldsymbol{r}_{t+1} \,\middle|\, S_t = s\right) p\left(\boldsymbol{R}_{t+2}^{N} \,\middle|\, S_{t+1} = s'\right) \\
&= \sum_{s'=0}^{N_s-1} \gamma_{t+1}(s, s')\beta_{t+1}(s')
\end{aligned} \tag{2.7}$$

where the equality between the second and third line comes from the Markov property. The values of β are obtained with a backward recursion on the trellis.

2.1.3 Computation of γ

From the definition of γ, we have

$$\begin{aligned}
\gamma_t(s', s) &= p\left(S_t = s,\, \boldsymbol{r}_t \,\middle|\, S_{t-1} = s'\right) \\
&= p\left(\boldsymbol{r}_t \,\middle|\, S_t = s,\, S_{t-1} = s'\right) \Pr\left(S_t = s \,\middle|\, S_{t-1} = s'\right),
\end{aligned} \tag{2.8}$$

where $\Pr\left(S_t = s \,\middle|\, S_{t-1} = s'\right)$ is the *a priori* probability of a transition between two states, defined by the structure of the trellis, and $p\left(\boldsymbol{r}_t \,\middle|\, S_t = s,\, S_{t-1} = s'\right)$ is the likelihood of $\boldsymbol{r}_t$ given a specific transition. It is dependent on how the channel affects the received vector $\boldsymbol{r}_t$.

2.1.4 APP of a Transition

Following the definition of α, β and γ, the APP of a transition can be expressed

$$\begin{aligned}\Pr\left(S_{t-1}=s',\, S_t=s \,\middle|\, \boldsymbol{R}_1^N\right) &= \frac{p\left(S_{t-1}=s',\, S_t=s,\, \boldsymbol{R}_1^N\right)}{p\left(\boldsymbol{R}_1^N\right)} \\ &= \frac{\alpha_{t-1}(s')\cdot\gamma_t(s',s)\cdot\beta_t(s)}{p\left(\boldsymbol{R}_1^N\right)}\end{aligned} \tag{2.9}$$

2.2 Application to Channel Coding

In channel coding, codewords are denoted $\boldsymbol{c}^i$, $i\in\{0,\ldots,M-1\}$, with M the size of the alphabet. Each codeword is composed of N_c binary values, and encodes an information word of length q (the relation between q and N_c may vary according to the selected code). Each codeword is mapped to one or multiple transitions of the trellis. We denote by $\mathcal{T}_i$ the set of transitions from s' to s which are mapped by the codeword $\boldsymbol{c}^i$. The APP of a codeword is given by the sum of the APP of the transitions which are mapped by the codewords, i.e.

$$\Pr\left(\boldsymbol{c}^i \,\middle|\, \boldsymbol{R}_1^N\right) = \sum_{\{s',s\}\in\mathcal{T}_i} \Pr\left(S_{t-1}=s',\, S_t=s \,\middle|\, \boldsymbol{R}_1^N\right). \tag{2.10}$$

2.2.1 Log-APP of the Information Bits

The information word corresponding to the time index t is denoted $\boldsymbol{b}_t$ and has q elements $b_{k,t}$, $k\in\{0,\ldots,q-1\}$. Each bit $b_{k,t}$ is equal to $u=\pm1$, and we denote by $\mathcal{B}_u^q$ the group of codewords that encode an information word for which the bit $b_k=u$ (the index t is dropped as the mapping between information words and codewords is the same for all time index). Considering the information bit at position $k=n$, the APP of this bit b_n

$$\begin{aligned}\Pr\left(b_{n,t}=u \,\middle|\, \boldsymbol{R}_1^N\right) &= \sum_{i\in\mathcal{B}_u^n} \Pr\left(\boldsymbol{c}^i \,\middle|\, \boldsymbol{R}_1^N\right) \\ &= \sum_{i\in\mathcal{B}_u^n}\sum_{\{s',s\}\in\mathcal{T}_i} \Pr\left(S_{t-1}=s',\, S_t=s \,\middle|\, \boldsymbol{R}_1^N\right) \\ &= \frac{1}{\Pr\left(\boldsymbol{R}_1^N\right)}\sum_{i\in\mathcal{B}_u^n}\sum_{\{s',s\}\in\mathcal{T}_i} \alpha_{t-1}(s')\cdot\gamma_t(s',s)\cdot\beta_t(s)\end{aligned} \tag{2.11}$$

The log-APP of the bit $b_{n,t}$ is given by

$$\begin{aligned}L\left(b_{n,t} \,\middle|\, \boldsymbol{R}_1^N\right) &= \log\frac{\Pr\left(b_{n,t}=+1 \,\middle|\, \boldsymbol{R}_1^N\right)}{\Pr\left(b_{n,t}=-1 \,\middle|\, \boldsymbol{R}_1^N\right)} \\ &= \log\frac{\displaystyle\sum_{i\in\mathcal{B}_{+1}^n}\sum_{\{s',s\}\in\mathcal{T}_i} \alpha_{t-1}(s')\cdot\gamma_t(s',s)\cdot\beta_t(s)}{\displaystyle\sum_{i\in\mathcal{B}_{-1}^n}\sum_{\{s',s\}\in\mathcal{T}_i} \alpha_{t-1}(s')\cdot\gamma_t(s',s)\cdot\beta_t(s)}.\end{aligned} \tag{2.12}$$

2.2.2 Computation of γ

The quantity $\gamma_t(s',s)$ is related to the transition probability, and can be expressed using the codeword $\boldsymbol{c}^i$ that maps the transition from state s' to s. The *a priori* probability of having that transition is the *a priori* probability to have the codeword $\boldsymbol{c}^i$ at time index t, yielding

$$\Pr\left(S_t=s \,\middle|\, S_{t-1}=s'\right) = \Pr\left(\boldsymbol{c}_t^i\right). \tag{2.13}$$

Likewise, the likelihood of the vector $\boldsymbol{r}_t$ given that the transition m' to m happened is expressed

$$p\left(\boldsymbol{r}_t \,\middle|\, S_t=s,\, S_{t-1}=s'\right) = p\left(\boldsymbol{r}_t \,\middle|\, \boldsymbol{c}_t^i\right). \tag{2.14}$$

The term $\gamma_t(s',s)$ can thus be expressed

$$\gamma_t(s',s) = p\left(\boldsymbol{r}_t \,\middle|\, \boldsymbol{c}_t^i\right) \Pr\left(\boldsymbol{c}_t^i\right), \tag{2.15}$$

giving an expression which includes the likelihood of the codeword $\boldsymbol{c}^i$ and the *a priori* probability of having the codeword. The likelihood can be computed under some assumptions on the channel, *e.g* an Additive White Gaussian Noise (AWGN) assumption.

By convention, the codeword $\boldsymbol{c}^i$ encodes an information word $\boldsymbol{b}^i$ of length q. The *a priori* probability of having the codeword $\boldsymbol{c}^i$ is expressed

$$\Pr\left(\boldsymbol{c}^i\right) = \prod_{k=0}^{q-1} \Pr\left(b_k = b_k^i\right) \tag{2.16}$$

where b_k refers to the value of the bit at index k. The log-ratio of the *a priori* probability of bit b_k is given by

$$L_A(b_k) = \log \frac{\Pr\left(b_k = +1\right)}{\Pr\left(b_k = -1\right)}. \tag{2.17}$$

Given that $\Pr\left(b_k = +1\right) + \Pr\left(b_k = -1\right) = 1$, we have

$$\Pr\left(b_k = +1\right) = \frac{e^{L_A(b_k)}}{1 + e^{L_A(b_k)}} = \frac{e^{-\frac{1}{2}L_A(b_k)} e^{\frac{3}{2}L_A(b_k)}}{\left(1 + e^{-L_A(b_k)}\right) e^{L_A(b_k)}} = \frac{e^{-L_A(b_k)/2}}{1 + e^{-L_A(b_k)}} e^{L_A(b_k)/2}, \tag{2.18}$$

which can be generalized to the two values $u = \pm 1$ using

$$\Pr\left(b_k = u\right) = \vartheta(b_k)\, e^{L_A(b_k)u/2}, \tag{2.19}$$

with

$$\vartheta(b_k) = \frac{e^{-L_A(b_k)/2}}{1 + e^{-L_A(b_k)}}. \tag{2.20}$$

The *a priori* probability of the codeword can thus be expressed

$$\begin{aligned} \Pr\left(\boldsymbol{c}^i\right) &= \prod_{k=0}^{q-1} \vartheta(b_k)\, e^{L_A(b_k) b_k^i/2} \\ &= \left(\prod_{n=0}^{\log_2(M)-1} \vartheta(b_k)\right) \exp\left\{\frac{1}{2} \sum_{n=0}^{\log_2(M)-1} L_A(b_k) b_k^i\right\} \end{aligned} \tag{2.21}$$

The likelihood of Equation (2.15) can be computed as

$$p\left(\boldsymbol{r}_t \,\middle|\, \boldsymbol{c}_t^i\right) = \prod_{k=0}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t}^i\right), \tag{2.22}$$

which gives a general expression for the terms γ_t as

$$\gamma_t(s',s) = \left(\prod_{k=0}^{q-1} \vartheta(b_{k,t})\right) \prod_{k=0}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t}^i\right) \exp\left\{\frac{1}{2} \sum_{k=0}^{q-1} L_A(b_{k,t}) b_{k,t}^i\right\}. \tag{2.23}$$

When considering binary codewords, the likelihood of a codeword can be expressed, using Equation (2.19)

$$p\left(\boldsymbol{r} \,\middle|\, \boldsymbol{c}^i\right) = \left(\prod_{k=0}^{N_c-1} \vartheta(c_k)\right) \exp\left\{\frac{1}{2} \sum_{k=0}^{N_c-1} L(c_k) c_k^i\right\}, \tag{2.24}$$

where $L(c_k)$ is the LLR of the bit c_k defined as

$$L(c_k) = \log \frac{p\left(r_k \mid c_k = +1\right)}{p\left(r_k \mid c_k = -1\right)}. \tag{2.25}$$

Using (2.21) and (2.24) in (2.15), it comes

$$\gamma_t(s',s) = \mathrm{C_{st}} \exp\left\{ \frac{1}{2} \sum_{k=0}^{N_c-1} L(c_{k,t}) c_{k,t}^i + \frac{1}{2} \sum_{k=0}^{q-1} L_A(b_{k,t}) b_{k,t}^i \right\} \tag{2.26}$$

with $\mathrm{C_{st}}$ a constant independent of i which will be cancelled out in further computations. If the code is systematic, then some positions of the codeword $\boldsymbol{c}^i$ correspond to the information word $\boldsymbol{b}^i$. The q systematic positions are denoted by $\mathcal{S}$, and the $N_c - q$ parity positions by $\mathcal{P}$. Equation (2.26) is expressed

$$\gamma_t(s',s) = \mathrm{C_{st}} \exp\left\{ \frac{1}{2} \sum_{k\in\mathcal{S}} \left((L(b_{k,t}) + L_A(b_{k,t})) b_{k,t}^i \right) + \frac{1}{2} \sum_{k\in\mathcal{P}} L(c_{k,t}) c_{k,t}^i \right\}. \tag{2.27}$$

In order to compute γ_t, the *a priori* log ratios are added to the LLR at the systematic positions.

2.2.3 Extrinsic Information

For the case of systematic binary codewords, the extrinsic information generated by the BCJR can be expressed as part of the log-APP of the bit. When considering one bit b_n, i.e. the bit at index $q = n$, the expression of $\gamma_k(s',s)$ can be factorized as

$$\begin{aligned}
\gamma_t(s',s) &= \prod_{k=0}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t} = c_{k,t}^i\right) \prod_{k=0}^{q-1} \Pr\left(b_{k,t} = b_{k,t}^i\right) \\
&= p\left(r_{n,t} \,\middle|\, c_{n,t} = b_{n,t}^i\right) \Pr\left(b_{n,t} = b_{n,t}^i\right) \prod_{\substack{k=0 \\ k\neq n}}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t} = c_{k,t}^i\right) \prod_{\substack{k=0 \\ k\neq n}}^{q-1} \Pr\left(b_{k,t} = b_{k,t}^i\right)
\end{aligned} \tag{2.28}$$

In the computation of the log-APP Equation (2.12), the top sum corresponds to the transitions for which the bit $b_{n,t}$ is equal to 1 and the bottom sum to the transitions for which the bit $b_{n,t}$ is -1. Using the previous factorization, it is thus possible to factorize the log-APP as

$$\begin{aligned}
L\left(b_{n,t} \,\middle|\, \boldsymbol{R}_1^N\right) = \log \frac{p\left(r_{n,t} \mid b_{n,t} = +1\right)}{p\left(r_{n,t} \mid b_{n,t} = -1\right)} + \log \frac{\Pr\left(b_{n,t} = +1\right)}{\Pr\left(b_{n,t} = -1\right)} \\
+ \log \frac{\displaystyle\sum_{i\in\mathcal{B}_{+1}^n} \sum_{\{s',s\}\in\mathcal{T}_i} \alpha_{t-1}(s') \cdot \prod_{\substack{k=0 \\ k\neq n}}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t} = c_{k,t}^i\right) \cdot \prod_{\substack{k=0 \\ k\neq n}}^{q-1} \Pr\left(b_{k,t} = b_{k,t}^i\right) \cdot \beta_t(s)}{\displaystyle\sum_{i\in\mathcal{B}_{-1}^n} \sum_{\{s',s\}\in\mathcal{T}_i} \alpha_{t-1}(s') \cdot \prod_{\substack{k=0 \\ k\neq n}}^{N_c-1} p\left(r_{k,t} \,\middle|\, c_{k,t} = c_{k,t}^i\right) \cdot \prod_{\substack{k=0 \\ k\neq n}}^{q-1} \Pr\left(b_{k,t} = b_{k,t}^i\right) \cdot \beta_t(s)}.
\end{aligned} \tag{2.29}$$

Finally, the log-APP can be expressed as the sum of three log ratios, with

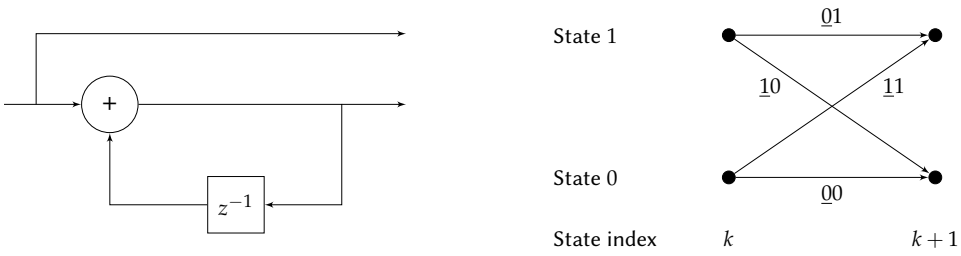
$$L\left(b_{n,t} \,\middle|\, \boldsymbol{R}_1^N\right) = L\left(b_{n,t}\right) + L_A\left(b_{n,t}\right) + L_E, \tag{2.30}$$

where L is the LLR, L_A the *a priori* log ratio, and L_E the extrinsic log ratio generated by the decoding process.

2.2.4 The BCJR Algorithm

The computation of the BCJR algorithm can be summarized in 4 steps

- $\alpha_0(s)$ and $\beta_N(s)$ are initialized according to the condition $S_0 = S_N = 0$.
- When the noisy codewords $\boldsymbol{r}_t$ are received, the transition probabilities $\gamma_t(s',s)$ are computed with (2.8), along with the forward recursion $\alpha_t(s)$ with (2.6). The computation of $\gamma_t(s',s)$ can be computed with Equation (2.23) or Equation (2.27) for the case of systematic binary codewords.
- Once the whole sequence $\boldsymbol{R}_1^N$ has been received, all the values of α and γ are known, and the backward recursion computes the values of β using (2.7).
- Once the values of α, β and γ are computed, the log-APP of the information bits can finally be computed with (2.12).



(a) (b)

FIGURE 2.1 – *Encoder (a) and trellis (b) of the accumulator.*

2.2.5 Example of the $[3\,1]$ Convolutional Code

The $[3\,1]$ RSC code is also known as the accumulator. Its encoder and trellis are depicted in the Figure 2.1. The trellis has 4 transitions, and each transition has a different codeword. There are $M = 4$ codewords of length $N_c = 2$ with $\boldsymbol{c}^i = \left[x^i,\, y^i\right]$, where x is the systematic bit and y the parity bit. There is only one systematic bit ($q = 1$) in each codeword. The received codeword at time t is $\boldsymbol{r}_t = [x_t,\, y_t]$.

2.2.5.1 Computation of γ

The value of γ is given by Equation (2.27) for the case of a binary systematic code. As the 4 codewords are associated to the 4 transitions, the usual expression $\gamma_t(s', s)$ can be simplified to $\gamma_t(\boldsymbol{c}^i)$. For this code, γ is computed as

$$\gamma_t(\boldsymbol{c}^i) = C \exp\left\{\frac{1}{2}(L(x_t) + L_A(x_t))x^i + \frac{1}{2}L(y_t)y^i\right\}, \tag{2.31}$$

where $x^i,\ y^i$ are binary values (equal to ± 1). If no *a priori* information is available, then $L_A(x_t) = 0$.

2.2.5.2 Computation of α and β

Following the recursive definition of α given Equation (2.6), and given the possible trellis transitions, the value of α at time index t is

$$\begin{aligned}\alpha_t(0) &= \alpha_{t-1}(0) \cdot \gamma_t([00]) + \alpha_{t-1}(1) \cdot \gamma_t([10]) \\ \alpha_t(1) &= \alpha_{t-1}(0) \cdot \gamma_t([11]) + \alpha_{t-1}(1) \cdot \gamma_t([01])\end{aligned}$$

Likewise, the computation of β, given Equation (2.7), is given by

$$\begin{aligned}\beta_t(0) &= \beta_{t+1}(0) \cdot \gamma_t([00]) + \beta_{t+1}(1) \cdot \gamma_t([11]) \\ \beta_t(1) &= \beta_{t+1}(0) \cdot \gamma_t([11]) + \beta_{t+1}(1) \cdot \gamma_t([01])\end{aligned}$$

CHAPTER 3

Extrinsic Information Transfer Chart

The EXIT chart was invented by Stephan ten Brink [3] to track the exchange of information inside a turbo receiver, and to predict the behavior of the constituent decoders [6]. In this document, the method and principle of the EXIT chart is reviewed. The [13 15] TC is used as an example for the interpretation of the EXIT chart tool and the extension for the multi-dimensional TC is presented.

3.1 Metric and Model

The EXIT analysis is performed by simulating the response of the decoder to an *a priori* input, while considering a noisy observation of the encoded bits. The inputs and output log ratios of the decoder are represented in Figure 3.1. The LLRs are obtained using the AWGN channel model with a certain noise level. A specific model is considered for the *a priori* input. The value of the parameter of the model simulates a strong *a priori* knowledge or, on the contrary, a poor *a priori* knowledge. In order to quantify the dependency between the *a priori* information fed to the decoder and the actual information, [3] suggested the use of the Mutual Information (MI).

In this section, the definition and expression of the MI is reviewed, and the model for the *a priori* information proposed in [3] is detailed, along with the expression of the extrinsic information.

3.1.1 The Mutual Information

The MI is a measure of the statistical dependency between two random variables. It can be defined using the Kullbach-Leiber divergence [12]

$$I(X, Y) = \int_x \int_y p(x, y) \log_2 \frac{p(x, y)}{p(x)\, p(y)} dx\, dy, \tag{3.1}$$

where $p(x)$ (resp. $p(y)$) is the probability of the event $X = x$ (resp. $Y = y$) and $p(x, y)$ the joint probability of the events. In the case where the two events are independent, we have $p(x, y) = p(x)\, p(y)$ and the MI is equal to 0. Otherwise, the MI is superior to 0.

With $p(x, y) = p(y \mid x)\, p(x)$, the MI is expressed

$$I(X, Y) = \int_x \int_{-\infty}^{+\infty} p(y \mid x)\, p(x) \log_2 \frac{p(y \mid x)}{p(y)} dx\, dy. \tag{3.2}$$

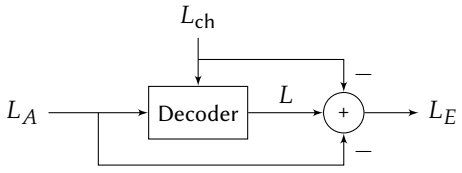


Figure 3.1 – *Inputs and output* log*-ratios of a decoder in a turbo receiver.*

Further considering the random variable X as the information source, whose values are binary ($X \in \{-1, +1\}$) and equally probable, the probability of Y can be obtained by marginalizing the joint probability over the possible values of X. This gives

$$\begin{aligned} p(y) &= \int p(x, y)\, dx \\ &= \sum_{u \in \{-1, +1\}} p(y \mid x = u) \Pr(x = u) \\ &= \frac{1}{2} \left(p(y \mid x = -1) + p(y \mid x = +1) \right), \end{aligned} \tag{3.3}$$

as the binary values of X are equally probable.

Considering the range of the observation y to be from $-\infty$ to $+\infty$, Equation (3.1) can now be expressed as

$$I(X, Y) = \frac{1}{2} \sum_{u \in \{-1, +1\}} \int_y p(y \mid x = u) \log_2 \frac{2p(y \mid x = u)}{p(y \mid x = -1) + p(y \mid x = +1)} dy \tag{3.4}$$

3.1.2 *A priori* Model

In order to model the *a priori* information fed to the decoder, [3] suggested the AWGN model, based on the observation of the distribution of the *a priori* log-ratios in a turbo receiver. To develop the model, we define the variable a as

$$a = x + n, \tag{3.5}$$

where x is the information (with $x = \pm 1$) and $n \sim \mathcal{N}(0, \sigma^2)$. Using the definition of the normal distribution, the log ratio is given by

$$\begin{aligned} L_A(x) &= \frac{p(a \mid x = +1)}{p(a \mid x = -1)} \\ &= \frac{2}{\sigma^2} a \\ &= \mu_A x + n_A, \end{aligned} \tag{3.6}$$

with

$$\mu_A = \frac{2}{\sigma^2}, \tag{3.7}$$

and

$$n_A = \frac{2}{\sigma^2} n. \tag{3.8}$$

Similarly, the log ratio L_A can be said to follow a normal distribution $\mathcal{N}(\mu_A, \sigma_A)$, where

$$\sigma_A^2 = \mathrm{Var}(n_A) = \mathrm{Var}\left(\frac{2}{\sigma^2} n \right) = \frac{4}{\sigma^4} \cdot \sigma^2 = \frac{4}{\sigma^2}. \tag{3.9}$$

Using the definition of μ_A and σ_A, this gives

$$\mu_A = \frac{\sigma_A^2}{2} \tag{3.10}$$

3.1.3 Expression of the *a priori* Mutual Information

While the standard definition of the MI requires an estimation of PDFs and the computation of several integrals, its computation can be simplified using the model selected for the *a priori* log ratios. The derivation of the simplified expression of the *a priori* MI is presented here.

Denoting $y = L_A(x)$, the distribution of the *a priori* log ratio is given by

$$p_A(y \mid x) = \frac{1}{\sqrt{2\pi}\sigma_A} \exp\left\{ -\frac{\left(y - \frac{\sigma_A^2}{2} x \right)^2}{2\sigma_A^2} \right\} \tag{3.11}$$

The computation of the *a priori* MI can be expressed

$$I_A = \frac{1}{2} \sum_{u \in \{-1,+1\}} \int_{-\infty}^{+\infty} p_A(y \mid x = u) \log_2 (2 f(y,u))\, dy \tag{3.12}$$

with

$$\begin{aligned} f(y,u) &= \frac{p_A(y \mid x = u)}{p_A(y \mid x = -1) + p_A(y \mid x = +1)} \\ &= \frac{p_A(y \mid x = u)}{p_A(y \mid x = +1)} \cdot \frac{1}{1 + \frac{p_A(y \mid x=-1)}{p_A(y \mid x=+1)}}. \end{aligned} \tag{3.13}$$

Using (3.11), it is easily shown that

$$\frac{p_A(y \mid x = -1)}{p_A(y \mid x = +1)} = e^{-y}, \tag{3.14}$$

which is often referred to as the consistency property of the normal distribution [13]. Likewise, the first term can be expressed

$$\begin{aligned} \frac{p_A(y \mid x = u)}{p_A(y \mid x = +1)} &= \frac{\exp\left\{-\frac{1}{2\sigma_A^2}\left(y^2 - \sigma_A^2 uy + \frac{\sigma_A^4}{4}u^2\right)\right\}}{\exp\left\{-\frac{1}{2\sigma_A^2}\left(y^2 - \sigma_A^2 y + \frac{\sigma_A^4}{4}\right)\right\}} \\ &= \exp\left\{\frac{1}{2}yu - \frac{1}{2}y - \frac{\sigma_A^2}{8}u^2 + \frac{\sigma_A^2}{8}\right\} && (3.15) \\ &= \exp\left\{\frac{1}{2}y(u-1)\right\}, && (3.16) \end{aligned}$$

where the developement from (3.15) to (3.16) comes from the fact that $u^2 = 1$ since $u = \pm 1$. Equation (3.13) can be expressed

$$f(y,u) = \exp\left\{\frac{1}{2}y(u-1)\right\} \cdot \frac{1}{1+e^{-y}}, \tag{3.17}$$

and the MI is expressed

$$I_A = \underbrace{\frac{1}{2}\int_{-\infty}^{+\infty} p_A(y \mid x = +1) \log_2\left(\frac{2}{1+e^{-y}}\right) dy}_{=I_1} + \underbrace{\frac{1}{2}\int_{-\infty}^{+\infty} p_A(y \mid x = -1) \log_2\left(\frac{2}{1+e^{y}}\right) dy}_{=I_2}. \tag{3.18}$$

The symmetry of the normal distribution gives $p_A(y \mid x = -1) = p_A(-y \mid x = +1)$, and the second integral I_2 is equal to

$$I_2 = \frac{1}{2}\int_{-\infty}^{+\infty} p_A(-y \mid x = +1) \log_2\left(\frac{2}{1+e^{y}}\right) dy. \tag{3.19}$$

By doing the variable change $z = -y$, the integral becomes

$$\begin{aligned} I_2 &= \frac{1}{2}\int_{+\infty}^{-\infty} p_A(z \mid x = +1) \log_2\left(\frac{2}{1+e^{-z}}\right)(-dz) && (3.20) \\ &= \frac{1}{2}\int_{-\infty}^{+\infty} p_A(z \mid x = +1) \log_2\left(\frac{2}{1+e^{-z}}\right) dz, && (3.21) \end{aligned}$$

which gives $I_1 = I_2$, and the MI expression becomes

$$\begin{aligned} I_A &= \int_{-\infty}^{+\infty} p_A(z \mid x = +1) \log_2\left(\frac{2}{1+e^{-z}}\right) dz && (3.22) \\ &= \int_{-\infty}^{+\infty} p_A(z \mid x = +1)\left(1 - \log_2\left(1+e^{-z}\right)\right) dz && (3.23) \\ &= 1 - \int_{-\infty}^{+\infty} p_A(z \mid x = +1) \log_2\left(1+e^{-z}\right) dz, && (3.24) \end{aligned}$$

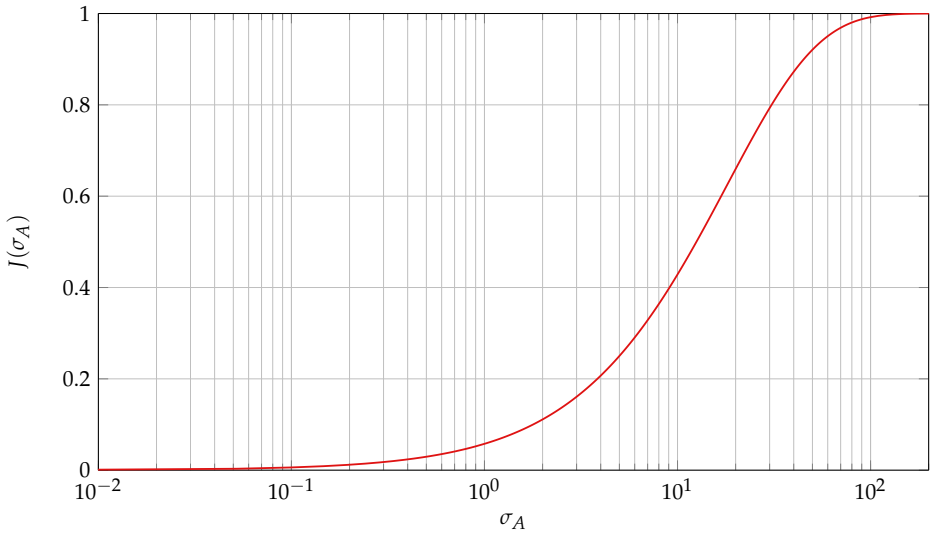


FIGURE 3.2 – *The J function.*

as the PDF's integral on the range of z is equal to 1. Including (3.11), the MI is computed with

$$I_A = I_A(\sigma_A) = 1 - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_A} \exp\left\{-\frac{1}{2\sigma_A^2}\left(z - \frac{\sigma_A^2}{2}\right)^2\right\} \log_2\left(1 + e^{-z}\right) dz, \tag{3.25}$$

which depends only on the parameter σ_A. The function J is usually defined with

$$J(\sigma) = I_A(\sigma_A = \sigma), \tag{3.26}$$

and represented in Figure 3.2 for various values of σ_A.

3.1.4 Extrinsic Information

When considering a turbo receiver, the two decoders exchanges extrinsic information. The extrinsic information of the first decoder is used by the second decoder as *a priori* input, and then its extrinsic information is sent back to the first decoder to be used as *a priori*, and so on all along the iterations. While the BCJR algorithm computes the log-APP of the information bits L, as defined by Equation (1.1), the extrinsic information L_E is obtained with

$$L_E = L - (L_A + L_{\text{ch}}). \tag{3.27}$$

In order to compute the dependency of the extrinsic information generated by the decoder and the information source, the MI is also used. Its value is computed using Equation (3.4), which requires the evaluation of the different PDFs and their integration.

3.2 The EXIT Chart

Using the MI metric and the model described previously, the computation method of the EXIT chart and its interpretation are presented. As the EXIT Chart computation does not depend on the interleaving function, this implies statistical independence between the two stages of the decoder. Additionally, the computations need to be done using very large block sizes Q, in order to ensure a sufficient statistic for the PDF estimations.

3.2.1 Computation

The computation of the EXIT is presented in Figure 3.3. The top part consists of a selected value of σ_A, with the application of the J function of Equation (3.26) to compute the *a priori* MI and uses σ_A to generate the *a priori* log ratios. The bottom part consists of the simulation of the transition through the channel. The AWGN channel is selected and after encoding of the information bits, the LLRs from the channel are computed. The simulated *a priori* log ratios are then added to the systematic bits, and the BCJR algorithm is applied to compute the log-APP of

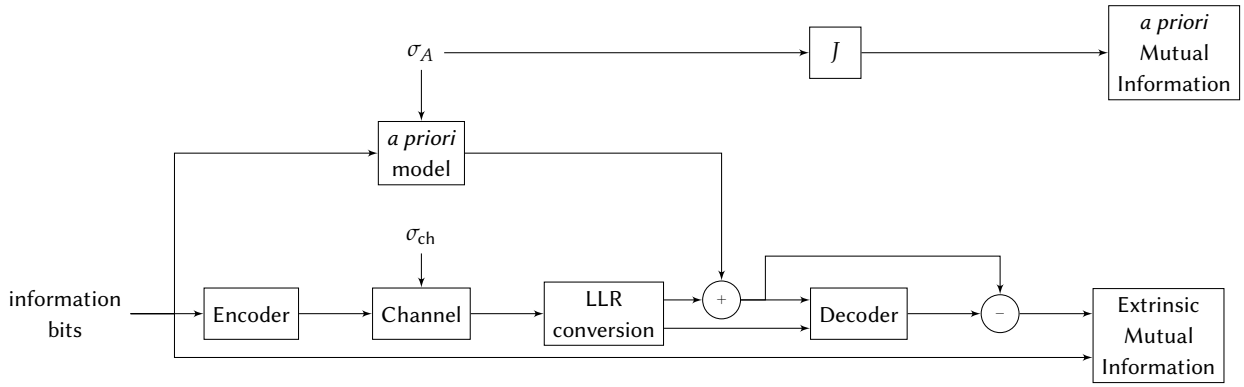


FIGURE 3.3 – *Computation of the EXIT chart.*

the information bits. The extrinsic log ratios are obtained using Equation (3.27), and the extrinsic MI is computed using Equation (3.4), which requires the knowledge of the information bits to estimate the conditional PDFs.

Both the *a priori* and the extrinsic MIs are estimated for various values of σ_A for a given σ_{ch}. Usually, the values of σ_A are chosen for the *a priori* MI I_A range from 0 to 1.

3.2.2 INTERPRETATION

In order to illustrate the interpretation of the EXIT chart tool, the RSC with generators [13 15] is chosen. Its encoder and trellis are represented on the Figure 1.1. This code has a rate $R = 1/3$, and in the turbo receiver (depicted Figure 1.2 (b)), the extrinsic of one decoder is used by the other decoder as *a priori*, and vice versa.

In Figure 3.4, the EXIT chart of the considered code is represented for various values of the channel parameter E_b/N_0. The *a priori* MI I_A spans from 0 (no *a priori* information) to 1 (complete knowledge of the information bits). The extrinsic MI I_E evolves differently with I_A depending on the value of E_b/N_0, and accurately reaches the coordinate $(1, 1)$ (i.e. with a perfect *a priori* knowledge, the correct information word is decoded). However, this curve only represents one of the two constituents of the turbo receiver. The other decoder, in the case where both constituent codes are the same, will have the same EXIT curve. [3] shows that in this case, the EXIT should be compared to the diagonal line going from $(0, 0)$ towards $(1, 1)$. If there is an intersection, the receiver will not be able to decode the correct information word, whatever the number of iterations. From the results depicted in Figure 3.4, where the diagonal line is in dashed red, we can conclude that the receiver cannot decode correctly the information word for E_b/N_0 values of -1 and -0.1dB, but can for values above that.

In order to illustrate and clarify this, the EXIT chart of the two constituant decoders are represented in Figure 3.5 for $E_b/N_0 = 1$dB, but the second one is turned over. Its *a priori* MI is on the y axis and its extrinsic MI is on

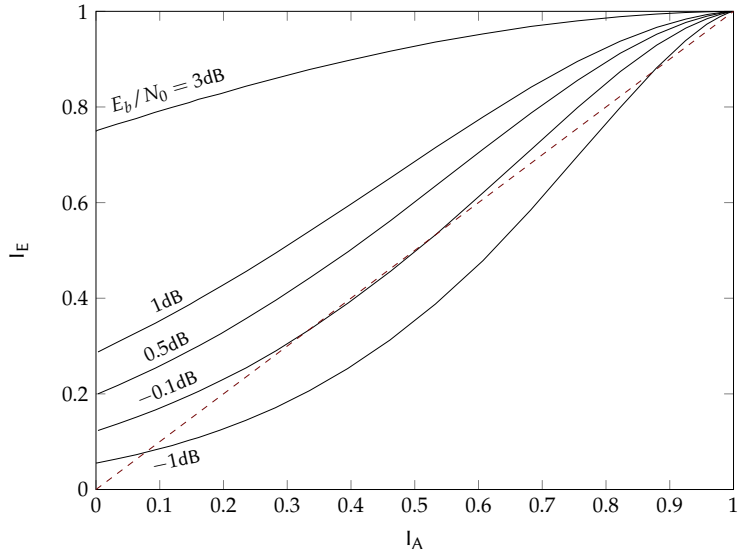


FIGURE 3.4 – *EXIT charts of the* [13 15] *code for multiple values of* E_b/N_0 *and the diagonal line. The information block size is set to* $Q = 100\,000$.

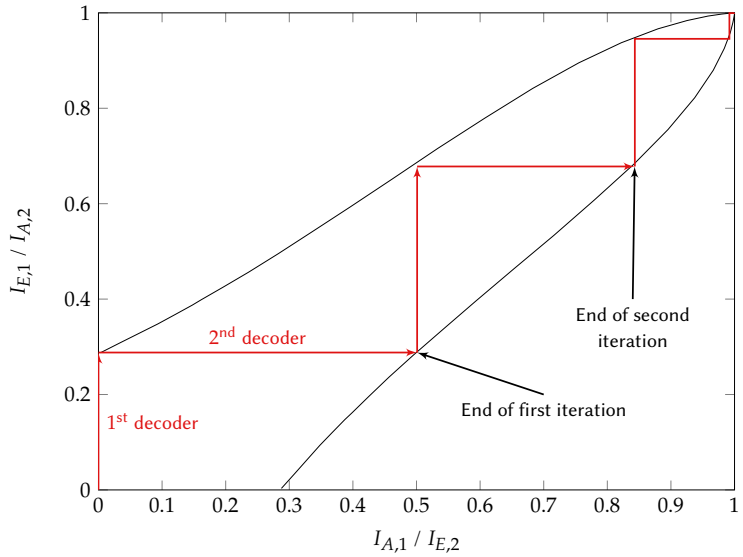


FIGURE 3.5 – *One trajectory of the exchanges of information inside the receiver of the* [13 15] *TC, for* $E_b/N_0 = 1dB$. *The information block size is set to* $Q = 100\,000$.

the x axis. This way, the exchanges of extrinsic information during the decoding process, or the "trajectory" can be directly visualized:

- The system starts at the point $(0,\ 0)$. The first decoder $I_{A,1}$ is 0, and outputs the value $I_{E,1} \simeq 0.3$.
- The second decoder has for input $I_{A,2} = I_{E,1} \simeq 0.3$, and reading the reversed curve, its output is $I_{E,2} \simeq 0.5$. As the two decoders performed one estimation, this is the end of the first iteration.
- The same process is repeated to read the value of $I_{E,2}$ at the end of the second iteration.
- After roughly 4 iterations, the coordinates $(1,\ 1)$ are reached: the correct word will be decoded.

The trajectory was not drawn on the plot, but actually computed, showing that the EXIT chart predicted accurately the behavior of the system. Due to the symmetry of the system, the comparison to the diagonal line is sufficient, as previously mentioned.

In Figure 3.6, the trajectories of the receiver for the values of E_b/N_0 of 0dB (a) and -1dB (b) are represented. For (a), the receiver converges toward the coordinates $(1,\ 1)$, but after significantly more iterations ($\simeq 9$) than in the case $E_b/N_0 = 1$dB presented in Figure 3.5. On the opposite, the receiver for $E_b/N_0 = -1$dB converges towards the coordinates $(0.08,\ 0.08)$, showing that the correct information word cannot be retrieved for the value of E_b/N_0.

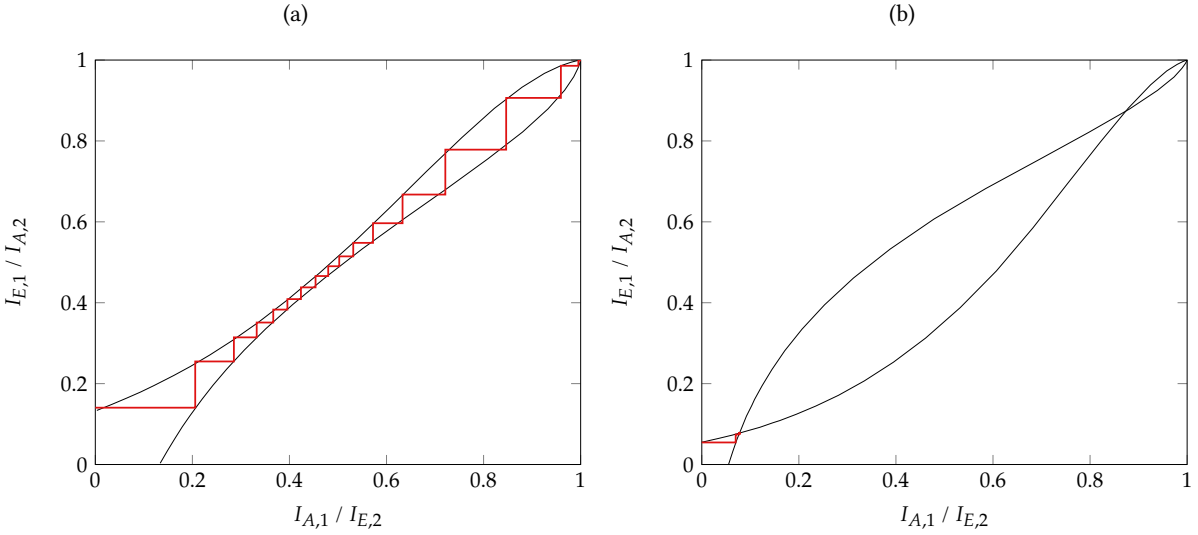


FIGURE 3.6 – *One trajectory of the exchanges of information inside the receiver of the* [13 15] *TC, for (a)* $E_b/N_0 = 0dB$, *(b)* $E_b/N_0 = -1dB$. *The information block size is set to* $Q = 100\,000$.

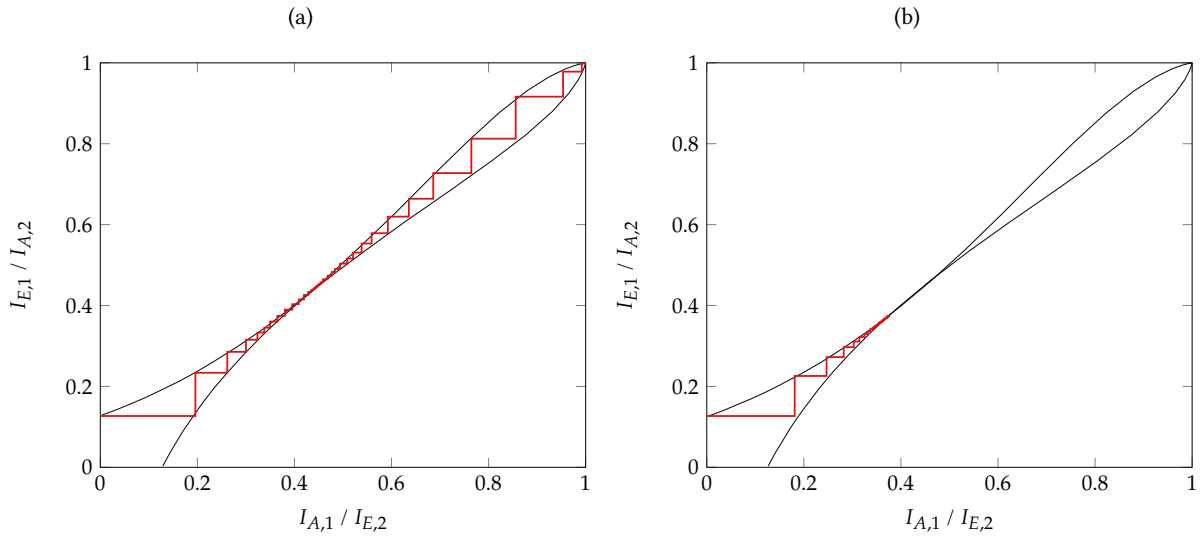


FIGURE 3.7 – *One trajectory of the exchanges of information inside the receiver of the* [13 15] *TC, for (a)* $E_b/N_0 = -0.05$*dB, (b)* $E_b/N_0 = -0.07$*dB. The information block size is set to* $Q = 100\,000$.

Since the EXIT chart accurately predicts the behavior of the turbo receiver, it can also be used to estimate the threshold of the decoder (also called the turbo-cliff), i.e. the E_b/N_0 value from which the decoder can retrieve the information word correctly after an arbitrary number of iterations. From the results previously presented, this is the value of E_b/N_0 for which the EXIT curve does not intersect the diagonal line. In Figure 3.7, the EXIT charts and one trajectory of the receiver for the values of E_b/N_0 of -0.05 and -0.07dB are represented. Since there is no intersection for the case $E_b/N_0 = -0.05$dB (the trajectory reaches $(1, 1)$) and intersection for the case $E_b/N_0 = 0.07$dB (the trajectory stops around $(0.35, 0.35)$), we can conclude that the threshold is between the two values of E_b/N_0.

It is important to note that the estimation of the threshold is done for very large values of information block size Q, and does not indicate the number of iterations required for the receiver to converge toward the maximum of MI. This is an asymptotic performance, i.e. a lower bound on the performance of the code.

3.3 Multi-dimensional EXIT Chart

While the initial TC [1] consists of two codes concatenated in parallel, the use of more than two constituent codes has been suggested [14]. Following this idea, Stefan ten Brink proposed an extension of the EXIT chart analysis to multi-dimensional coding schemes [15]. In this type of receiver, the *a priori* information fed to one decoder consists of the sum of the extrinsic of the other decoders.

In the general case with λ parallel concatenated codes, the representation of the exchanges of information needs to be done with λ dimensions. This becomes inconvenient when $\lambda > 3$. However, in the specific case where all the codes are the same, the process can be simplified by computing a two dimensional projection. This projection is obtained by computing the EXIT chart of one decoder while considering that the *a priori* input consists in the sum of the extrinsic of the other decoders. This implies a correction in the computation of the *a priori* MI.

In order to express the correction, the more convenient case of $\lambda = 3$ is considered first. In this case, the *a priori* of the first decoder L_{A_1} (after at least one iteration) is given by

$$L_{A_1} = L_{E_2} + L_{E_3} \tag{3.28}$$

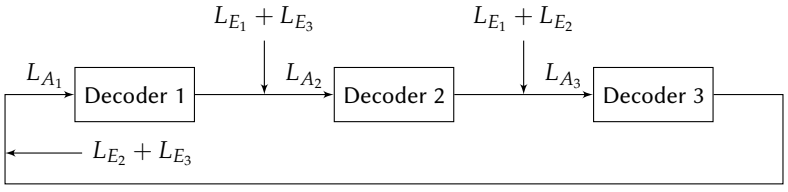


FIGURE 3.8 – *Illustration of the exchange of information between the three decoders.*

where L_{E_2} (resp. L_{E_3}) is the extrinsic log ratio of the second (resp. the third) decoder. The exchange of information in the three dimensions receiver is depicted on the Figure 3.8.

Assuming a Gaussian distribution for the log ratios and the independence of the extrinsic observations [15],

$$\sigma_{A_1}^2 = \sigma_{E_2}^2 + \sigma_{E_3}^2. \tag{3.29}$$

Using $I = J(\sigma)$, and inversely $\sigma = J^{-1}(I)$, the *a priori* MI is given by

$$\begin{aligned} I_{A_1} = J(\sigma_{A_1}) &= J\left(\sqrt{\sigma_{E_2}^2 + \sigma_{E_3}^2}\right) \\ &= J\left(\sqrt{\left(J^{-1}(I_{E_2})\right)^2 + \left(J^{-1}(I_{E_3})\right)^2}\right). \end{aligned} \tag{3.30}$$

By setting $I_{E_2} = I_{E_3}$, then

$$I_{A_1} = J\left(\sqrt{2} \cdot J^{-1}(I_{E_2})\right). \tag{3.31}$$

With the same *a priori* model with variance σ_A^2 described in Section 3.1.2, the *a priori* MI is thus given by

$$I_{A_1} = J\left(\sqrt{2} \cdot \sigma_A\right), \tag{3.32}$$

which can be generalized to λ decoders as

$$I_A = J\left(\sqrt{\lambda - 1} \cdot \sigma_A\right). \tag{3.33}$$

In order to obtain the two dimensions' projection, the *a priori* MI is computed with Equation (3.33), and for various values of σ_A, the extrinsic MI is computed. When the transition in the channel is modeled, the overall rate of the code is considered.

Chapter 4

The TurboCode class

In order to illustrate the concept described previously, a MATLAB class has been developed. Using the class and with specific parameters, various features can be analyzed. BER curves and EXIT charts can be computed, and trajectories of the decoder can be observed. In this chapter, basic instructions on how to access and use the class are presented, along with the main properties and methods of the class.

What does the code can do ?

The code proposed here can instantiate an object of type `TurboCode` with the specified parameters. BER and Packet Error Rate (PER) simulations can be performed, EXIT charts can be computed and trajectories of the decoder can be represented. The code only handles symmetric TC with two similar rate 1/2 RSC consistuents.

4.1 Accessing Matlab Material

The MATLAB code can be accessed through the git repository at https://github.com/yoannroth/turbocode.git. The repository can be cloned locally using

```
git clone https://github.com/yoannroth/turbocode.git
```

The repository contains a folder "matlab" which includes the class, functions and various examples of possible scripts.

4.2 Properties of the Class and Constructors

The properties of the class correspond to the possible configuration for the turbo code (both encoder and receiver). The possible properties and their signification are listed below:

- `blkLength` : `int` value representing the size of the binary information word to be encoded. The default value is 1024.
- `poly` : `int` vector of size 2 containing the polynomials generators of the RSC consistuent. The default value is [13, 15].
- `k` : `int` value representing the constraint length of the code (defined as number of memory +1). This value should match the constraint length required for the generator specified with `poly`. The default value is 4.
- `puncturing` : `bool` value enabling the puncturing of half of the parity bits. The overall rate becomes close to 1/2. The default value is `false`.
- `algorithm` : `char` value specifying the algorithm to use for decoding the trellis. Two values are possible: `'MAP'` or `'maxlog'`. The default value is `'MAP'`.
- `numIter` : `int` value specifying the number of decoding iterations to process. The default value is 10.
- `backup` : `bool` value enabling the save of the simulation results. The default value is `true`.

The default constructor of the class, `TurboCode()`, instantiate an object using the default parameters described above. However, specific value for the parameters can be selected by calling

```
TurboCode('<paramName>', <paramValue>, ...)
```

where `<paramName>` is the name of the parameter and `<paramValue>` the desired value for the parameter. For example, instantiating an object `tc` for which the block length is set to 100 000 and with puncturing can be done with

```
tc = TurboCode('blkLength', 100000, 'puncturing', true);
```

The other parameters will take the default values.

Other properties, such as the rate of the code or an identifier string, are computed in the constructor and cannot be specified.

4.3 Public Methods

The class contains several public methods which execute various computations or plot some data. The methods are listed below:

- `[ber, per] = computeBer(ebno, varargin)` The method computes the BER (and the PER) for the specified values of E_b/N_0. Two optional arguments can be used: the number of errors required for the simulation of one E_b/N_0 value to stop and the BER under which the overall simulation will stop. The default values are set to 1000 and 10^{-6} (resp.).
- `[ia, ie] = computeExit(ebno, varargin)` The method computes the EXIT chart for the specified value of E_b/N_0. Two optional arguments can be used: the number simulation to run and the I_A values to test. The default values are set to 10 and $0 : 0.1 : 1$ (resp.).
- `[ia, ie] = computeTraj(ebno, varargin)` The method computes the trajectory of the decoder for the specified value of E_b/N_0. Two optional arguments can be used: the number of trajectories to simulate and if the result must be averaged over the trajectories. The default values are set to 1 and `false` (resp.).
- `h = plotBer(ebno, ber, varargin)` The method plots the BER versus E_b/N_0. Two optional arguments can be used: the flag `'snr'` to plot the BER versus the Signal-to-Noise Ratio (SNR), and the iteration indexes to plot (this can be a vector or scalar, set to `numIter` by default).
- `h = plotExit(ebno, ia, ie)` The method plots the EXIT chart for the given E_b/N_0.
- `h = plotTraj(ebno, ia, ie)` The method plots the trajectories for the given E_b/N_0 along with the EXIT chart. If not EXIT chart for this value of E_b/N_0 is found, the method `computeExit` is called to create one.

There are also several private methods computing internal steps of the algorithms. Also, the function `ForwardBackward` includes recursive steps and is thus implemented in `Cmex` for better computing performance.

List of Acronyms

APP	· *A Posteriori* Probability
AWGN	· Additive White Gaussian Noise
BCJR	· Bahl, Cocke, Jelinek and Raviv
BER	· Bit Error Rate
BPSK	· Binary Phase Shift Keying
CC	· Convolutional Code
EXIT	· EXtrinsic Information Transfer
LLR	· Log Likelihood Ratio
MAP	· Maximum *A Posteriori*
MI	· Mutual Information
PCCC	· Parallel Concatenated Convolutional Code
PDF	· Probability Density Function
PER	· Packet Error Rate
RSC	· Recursive Systematic Convolutional
SNR	· Signal-to-Noise Ratio
SOVA	· Soft-Output Viterbi Algorithm
TC	· Turbo Code

Bibliography

[1] C. Berrou, A Glavieux, and P. Thitimajshima. “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes”. In: *IEEE International Conference on Communications (ICC). Geneva.* Vol. 2. 1993, pp. 1064–1070.

[2] G. D. Forney. *Concatenated Codes.* Cambridge, 1966.

[3] S. ten Brink. “Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes”. In: *IEEE Transactions on Communications* 49.10 (2001), pp. 1727–1737.

[4] S. Benedetto and G. Montorsi. “Average Performance of Parallel Concatenated Block Codes”. In: *Electronics Letters* 31.3 (1995), pp. 156–158.

[5] L. C. Perez, J. Seghers, and D. J. Costello. “A Distance Spectrum Interpretation of Turbo Codes”. In: *IEEE Transactions on Information Theory* 42.6 (1996), pp. 1698–1709.

[6] M. El-Hajjar and L. Hanzo. “EXIT Charts for System Design and Analysis”. In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 127–153.

[7] J. Hagenauer and L. Papke. “Decoding "Turbo"-Codes with the Soft Output Viterbi Algorithm (SOVA)”. In: *Proceedings of 1994 IEEE International Symposium on Information Theory.* 1994, pp. 164–.

[8] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer. “On the Equivalence Between SOVA and max-log-MAP Decodings”. In: *IEEE Communications Letters* 2.5 (1998), pp. 137–139.

[9] P. Robertson, P. Hoeher, and E. Villebrun. “Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding”. In: *European Transactions on Telecommunications* 8.2 (1997), pp. 119–125.

[10] J. Vogt and A. Finger. “Improving the max-log-MAP Turbo Decoder”. In: *Electronics Letters* 36.23 (2000), pp. 1937–1939.

[11] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate (Corresp.)” In: *IEEE Transactions on Information Theory* 20.2 (1974), pp. 284–287.

[12] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. In: *Ann. Math. Statist.* 22.1 (Mar. 1951), pp. 79–86.

[13] F. Alberge. “On Some Properties of the Mutual Information Between Extrinsics With Application to Iterative Decoding”. In: *IEEE Transactions on Communications* 63.5 (2015), pp. 1541–1553.

[14] C. Berrou, M. Jezequel, and C. Douillard. “Multidimensional Turbo Codes”. In: *1999 Information Theory and Networking Workshop (Cat. No.99EX371).* 1999, pp. 27–.

[15] S. ten Brink. “Convergence of Multidimensional Iterative Decoding Schemes”. In: *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers.* Vol. 1. 2001, 270–274 vol.1.