# Chapter 4

# Performance Analysis Techniques

## 4.1 Introduction

Efficient system design requires the use of tools for evaluating the performance of a given system. Based on a performance analysis method, design algorithms can be constructed, for example, with a specific performance optimization goal. The ideal analysis tool should be (i) fast, (ii) accurate, and, possibly, (iii) highly informative, i.e., it should provide a comprehensive description of the system under analysis. However, exact analytical evaluation of the relevant statistical parameters characterizing a realistic communication system is, usually, unfeasible. Therefore, it is necessary to resort to approximate evaluation tools. A universal technique satisfying the requirements (ii) and (iii) is given by a Monte Carlo simulation-based analysis of the system performance. In particular, this simulation method allows to collect all needed system statistical parameters with the desired accuracy. Unfortunately, system simulation is usually a computationally intensive task. This makes this technique not appealing for automated system parameter optimization (or design), which usually calls for repeated analyses. Nevertheless, Monte Carlo simulation is an invaluable tool suited for accurately testing the performance of a complex system. Iterative receiver schemes can also be investigated through a number of approximate tools which exploit their internal structures. For example, *density evolution* and extrinsic information transfer (EXIT) charts admit efficient numerical implementation, since they are characterized by a limited computational complexity.

In this chapter, the main numerical performance analysis tools useful for

M. Franceschini, G. Ferrari and R. Raheli, *LDPC Coded Modulations,*
DOI: 10.1007/978-3-540-69457-1_4, © Springer-Verlag Berlin Heidelberg 2009

the analysis and design of low-density parity-check (LDPC) codes for coded modulations will be described. In Section 4.2, Monte Carlo techniques are introduced and some considerations on their correct implementation and use are drawn. In Section 4.3, the density evolution method for analysis of iterative decoders is described. In Section 4.4, EXIT charts are introduced. Section 4.6 concludes this chapter.

## 4.2 Monte Carlo System Simulation

By system simulation, we refer to the practice of reproducing a set of signals which are statistically equivalent to those found in the actual system under analysis. To this end, the system is decomposed into its component blocks. We assume that the system may be decomposed into a network of blocks characterized by deterministic behavior and driven by inputs which may be either deterministic or stochastic. This assumption is quite general and may be applied in almost all scenarios of interest.

The deterministic blocks of the system are reproduced by implementation of the corresponding (embedded) signal processing algorithms or by numerical solution of the input-output equations of their mathematical model. The stochastic input signals may be generated in two different ways.

The first is to implement a physical device comprising a controlled and precisely known noise source, such as, e.g., amplified thermal noise, which is used to obtain a signal whose statistics are similar to those of the stochastic input to the system. In this case, the fact that the noise source parameters are not perfectly known is a possible cause of mismatch between the simulated system and the actual one.

The second, and most relevant for our purposes, technique for generating the input stochastic signals is by means of a pseudo-random number generator (PRNG). PRNGs are recursive algorithms which may be viewed as autonomous systems whose initial state is referred to as *seed* and whose output, interpreted as the realization of an ergodic and stationary process, implies statistical properties of the process such as, e.g., independent and identically distributed (i.i.d.) outputs, uniform distribution of the output sample over the integers within a given range, etc. Obviously, since the output is actually a deterministic sequence, the statistical description of the sequence is only an approximation of the desired one. Nevertheless, the output sequence is plausible for the stochastic model that the PRNG tries to emulate. The "randomness" of the sequence can be evaluated using a number of statistical evaluation mathematical tools. In order to obtain stochastic processes with

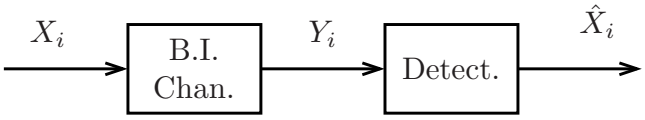


Figure 4.1: Schematic diagram of a uncoded binary-input (BI) simulator.

the desired statistical description, the output of the PRNG can be properly processed applying standard methods [56].

It is important to note that not every PRNG may be suited for the specific scope of a problem. On the other hand, in general, it is not necessary to use a true—as opposed to pseudo—random number generator. Every simulation scenario has its own characteristics. For example, almost every PRNG has a periodic behavior, i.e., the sequence repeats itself after a finite number of samples. Some applications may require a very long period in order to guarantee accurate estimation of the relevant statistical parameters. Other applocations, such as, for example, the generation of information bits in an uncoded binary-input (BI) memoryless channel simulator, can achieve the same or better results than those obtained with a true random number generator, with a PRNG with period as short as 2. This is considered in the following example.

**Example 4.1** *Simple PRNGs might be better than true RNGs*

We now compare the quality of BER estimate using a real random input data sequence and one generated using a PRNG with cycle of length 2. Showing that, even if the real communication system will operate on a random data sequence, the use of a PRNG leads to better estimate of the BER of that system. Consider a simulator for the uncoded transmission scheme in Figure 4.1, where the input sequence $\{X_i\}$ is a random sequence of i.i.d. bits $\{X_i\}$ with $P\{X_i = 1\} = P\{X_i = 0\} = 1/2$. The symbol $Y_i$ output by the BI channel at epoch $i$ depends only on $X_i$, i.e., the channel is memoryless. The sequence is detected by means of a detector device which outputs estimates of the transmitted bits on a bit-by-bit basis. For example, choose the maximum *a posteriori* probability (MAP) symbol decision rule, i.e., choose the bit $\hat{X}_i$ whose conditional a posteriori probability, given the received $Y_i$, is highest. We assume that the conditional probability of error $P_{e,0}$ associated with the transmission of a 0 might be different from the conditional probability of error $P_{e,1}$ associated with the transmission of a 1.

We use Monte Carlo simulation for measuning the bit error rate (BER) of the system. We generate $N$ bits to be transmitted $X_1, \ldots, X_N$ and send

them through the channel obtaining $N$ output samples $Y_1, \ldots, Y_N$. As BER estimator we consider the sample average of the error indicator function:

$$\hat{P}_e = \frac{1}{N} \sum_{i=1}^{N} 1\{\hat{X}_i \neq X_i\}$$

where the *indicator function* $1(\mathcal{A})$ of an event $\mathcal{A}$ is given by the following definition:

$$1(\mathcal{A}) = \begin{cases} 1 & \text{if } \mathcal{A} \text{ is true} \\ 0 & \text{else.} \end{cases}$$

In other words, $\sum_{i=1}^{N} 1\{\hat{X}_i \neq X_i\}$ is the total number of errors in $N$ trials.

We now characterize the variance of the BER estimator considering the two cases of a real random number generator and a PRNG.

First we measure probability of error using real random number generators. The bits to be transmitted are therefore $N$ i.i.d. equiprobable random bits $X_1, \ldots, X_N$. The mean of the estimator is

$$\mathrm{E}\{\hat{P}_e\} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{E}\{1\{\hat{X}_i \neq X_i\}\} = \frac{1}{N} \sum_{i=1}^{N} P_e = P_e$$

where the fact that $\mathrm{E}\{1(\mathcal{A})\} = \Pr\{\mathcal{A}\}$ has been used and $\Pr\{\mathcal{A}\}$ denotes the probability of event $\mathcal{A}$. As a consequence, the estimator is *unbiased* and its variance is

$$\begin{aligned} \mathrm{Var}\{\hat{P}_e\} &= \frac{1}{N^2} \sum_{i=1}^{N} \mathrm{Var}\{1\{\hat{X}_i \neq X_i\}\} \\ &= \frac{1}{N^2} \sum_{i=1}^{N} \mathrm{E}\{1^2\{\hat{X}_i \neq X_i\}\} - \mathrm{E}^2\{1\{\hat{X}_i \neq X_i\}\} \\ &= \frac{1}{N^2} \sum_{i=1}^{N} P_e - P_e^2 \\ &= \frac{P_e - P_e^2}{N} \end{aligned}$$

where $\mathrm{Var}\{\cdot\}$ denotes the variance of a random variable.

Assume now that we can accurately simulate the exact statistical channel distribution (e.g., by *using* the actual channel) but we send alternatively 1 and 0, i.e., $X_{2i-1} = 1$ and $X_{2i} = 0$, $i \geq 1$. This corresponds to the use of a particular PRNG with periodicity equal to 2 to generate the $\{X_i\}$ sequence.

As before, we generate $N$ (even) samples and compute the mean and variance of our estimator applied to the newly obtained sequence. Since it holds that

$$\begin{aligned}
\mathrm{E}\{\hat{P}_e\} &= \mathrm{E}\left\{\frac{1}{N}\sum_{i=1}^{N} 1\{\hat{X}_i \neq X_i\}\right\} \\
&= \mathrm{E}\left\{\frac{1}{N}\left[\sum_{i=0}^{N/2-1} 1\{\hat{X}_{2i+1} \neq X_{2i+1}\} + \sum_{i=1}^{N/2} 1\{\hat{X}_{2i+2} \neq X_{2i+2}\}\right]\right\} \\
&= \frac{1}{N}\left[\sum_{i=0}^{N/2-1} \mathrm{E}\{1\{\hat{X}_{2i+1} \neq X_{2i+1}\}|X_{2i+1}=1\}\right. \\
&\qquad \left. + \sum_{i=1}^{N/2} \mathrm{E}\{1\{\hat{X}_{2i+2} \neq X_{2i+2}\}|X_{2i+2}=0\}\right] \\
&= \frac{1}{N}\left[\frac{N}{2} P\{\hat{X}_{2i+1} \neq X_{2i+1}\}|X_{2i+1}=1\}\right. \\
&\qquad \left. + \frac{N}{2} P\{\hat{X}_{2i+2} \neq X_{2i+2}\}|X_{2i+2}=0\}\right] \\
&= \frac{1}{2}P_{e,1} + \frac{1}{2}P_{e,0} \\
&= P_e
\end{aligned}$$

it follows that the estimator is *unbiased*. Moreover, its variance is

$$\begin{aligned}
\mathrm{Var}\{\hat{P}_e\} &= \frac{1}{N^2}\sum_{i=1}^{N} \mathrm{Var}\{1\{\hat{X}_i \neq X_i\}\} \\
&= \frac{1}{N^2}\left[\sum_{i=0}^{N/2-1} \mathrm{Var}\{I\{\hat{X}_{2i+1} \neq X_{2i+1}\}\right. \\
&\qquad \left. + \sum_{i=0}^{N/2-1} \mathrm{Var}\{1\{\hat{X}_{2i+2} \neq X_{2i+2}\}\}\right] \\
&= \frac{1}{N^2}\left[\frac{N}{2}(P_{e,1} - P_{e,1}^2) + \frac{N}{2}(P_{e,0} - P_{e,0}^2)\right] \\
&= \frac{1}{N}\left(\frac{P_{e,1} + P_{e,0}}{2} - \frac{P_{e,1}^2 + P_{e,0}^2}{2}\right) \\
&\leq \frac{P_e - P_e^2}{N}
\end{aligned}$$

where $P_{e,1} = P\{\hat{X}_{2i+1} \neq X_{2i+1} | X_{2i+1} = 1\}$ and $P_{e,0} = P\{\hat{X}_{2i+2} \neq X_{2i+2} | X_{2i+2} = 0\}$ and the last passage is due to the fact that

$$P_e^2 = \left(\frac{P_{e,1} + P_{e,0}}{2}\right)^2 \leq \frac{P_{e,1}^2 + P_{e,0}^2}{2}\,.$$

This means that the resulting variance of the estimator is better than or equal to that obtained with random bits, i.e., using realistic data. It turns out that if the channel is known to be symmetric, i.e., if $P_{e,0} = P_{e,1}$, as they do in the case of the BI additive white Gaussian noise (BIAWGN) channel investigated in Example 1.1, the two variances coincide.

In performing a system analysis through Monte Carlo simulation, one should carefully consider which parameters are to be estimated. This is important, since a simulation, in general, produces a large amount of data. It is usually not possible nor efficient to completely store the produced data and, afterwards, proceed with a batch processing. Typically, depending on the particular statistical parameter to be evaluated, only a small fraction of the produced data has to be *stored*, although the whole produced data may sometimes need to be *processed*. As a simple but meaningful example, consider the transmission system analyzed in the above example. In order to estimate the BER, one could store the sequences $\{Y_i\}$, $\{X_i\}$, and $\{\hat{X}_i\}$, then perform batch processing on the sequences and compute both the estimate $\hat{P}_e$ and and an estimate of its variance. This would require a memory of size $O(N)$.[1] Obviously, the same can be done with a memory as small as $O(1)$. In fact, the processing could be done by updating an *error counter* on a sample-by-sample (or bit-by-bit) basis, i.e., if $\hat{X}_i \neq X_i$ then increase the *error counter* by one. The estimate is obtained by dividing the number of errors by $N$. Its variance can be estimated, in a similar way, using a single update variable.

One should always remember that every estimate based on the output of a simulation is, as a matter of fact, a *measure* which is subject to error. Assume to simulate the transmission of $N$ bits by a communication system. By comparing the decided bits at the output of the receiver with the transmitted ones, one discovers that there are $X$ errors. Without any other knowledge, the most reasonable estimate of the BER $\tilde{P}_{\rm e}$ characterizing the system is

$$\tilde{P}_{\rm e} = \frac{X}{N}\,.$$

[1] We say that a quantity $g(N)$ is "on the order of" $f(N)$ or $O(f(N))$, if the limit

$$\lim_{N\to\infty} \frac{f(N)}{g(N)}$$

exists and is greater than 0.

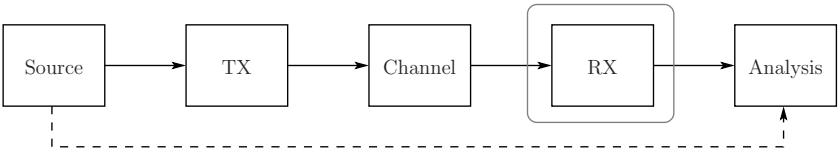


Figure 4.2: Block diagram of a Monte Carlo analysis system.

This estimator corresponds to the sample mean of the error indicator, and has good properties in several scenarios of practical interest. For example, if the communication system processes are stationary, it turns out that

$$\mathrm{E}\{\tilde{P}_\mathrm{e}\} = P_\mathrm{e}$$

where $P_\mathrm{e}$ is the true system BER. Note that every estimator is characterized by its own distribution. This allows to compute the uncertainties that should characterize every well done measure. For more details on this important topic we refer the reader to [56] and [57].

An important property of system simulation as analysis tool is that, if properly implemented, it is robust against possible implementation/programming errors. In Figure 4.2, a block diagram of a Monte Carlo simulation-based analysis system is shown. Each block, source, transmitter (TX), channel, receiver (RX), and statistical analysis, denote a "separate" software unit that may be implemented exploiting the most convenient paradigms of the adopted language or toolkit. By "separate" software unit we mean that it does not rely on nor may access other data besides those passed through the arrow connection, which carry only the sampled signal through the system. The receiver block is highlighted to emphasize that it is the block where the biggest design effort is spent. The other blocks, except for the transmitter which may make use of sophisticated signal processing algorithms, are usually very simple. If every other processing block but the receiver is known to be working correctly, a Monte Carlo simulation guarantees the following interesting robustness property: *if the receiver implementation is defective, i.e., it comprises one or more implementation errors, there cannot be false improvements of the system performance*. In other words, although defective, the implemented receiver algorithm can be considered as a real algorithm working on real data and the computed performance corresponds to the performance of a receiver using the buggy algorithm.

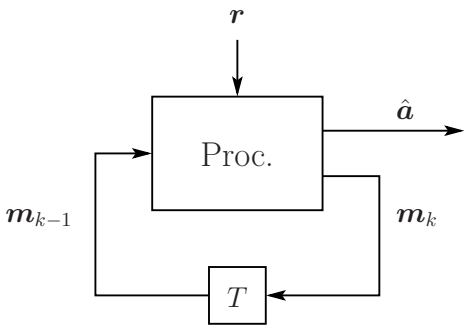


Figure 4.3: Illustrative diagram of a generic iterative detection scheme.

## 4.3 Density Evolution

Consider a generic iterative detection algorithm. The transmitted data sequence $\boldsymbol{a}$ produces a received vector $\boldsymbol{r}$ which is the input to the iterative detection scheme. A possible representation of a generic iterative algorithm is shown in the diagram in Figure 4.3. The receiver observes the vector $\boldsymbol{r}$ and generates a first vector of messages $\boldsymbol{m}_0$. The iterative process begins and at each iteration the previously obtained message set $\boldsymbol{m}_{k-1}$ is processed by a deterministic function which, on the basis of $\boldsymbol{m}_{k-1}$ and $\boldsymbol{r}$, computes the next message set $\boldsymbol{m}_k$. Note that, although the input vector $\boldsymbol{r}$ is random, the processing block operations are deterministic. At the end of the iterative process, e.g., after $\ell$ iterations, the output $\hat{\boldsymbol{a}}$ is computed as a function of the last message set $\boldsymbol{m}_\ell$ and of the input $\boldsymbol{r}$.

A complete statistical characterization of the iterative process would require to compute how the conditional joint probability density function (pdf) $p(\boldsymbol{m}_k|\boldsymbol{a})$ evolves as a function of $k$, and considering all possible transmitted data sequences $\boldsymbol{a}$. This is, however, impractical since the number of possible data sequences is an exponential function of the data sequence length and the size of the vector $\boldsymbol{m}_k$ is typically large. An LDPC code decoder would comprise a number of messages equal to the number of edges in the LDPC code's bipartite graph, which in practical applications, as a rule of thumb, may span from 2 to 6 times the codeword length. This implies that such an analysis of a practical LDPC code would require to compute the evolution of a joint pdf of about $10^4$-$10^5$ random variables (RVs).

A technique which can be used to effectively analyze iterative decoding schemes is the so-called *density evolution* [34, 58]. This technique is based

on a single-letter analysis of the evolution of the messages in an iterative decoding algorithm. In other words, the analysis focuses on the computation of the output distribution of a single message $m_k^{(i)}$. The message $m_k^{(i)}$ is a deterministic function of a subset $m_{k-1}^{(j_1)}, \ldots, m_{k-1}^{(j_{d_i})}$ of $d_i$ elements of the vector of messages $\boldsymbol{m}_{k-1}$ at the output of the processing block at the iteration $k-1$:

$$m_k^{(i)} = f_i(m_{k-1}^{(j_1)}, \ldots, m_{k-1}^{(j_{d_i})}) \,. \tag{4.1}$$

The pdf of $m_k^{(i)}$ is a function of the joint distribution of the input messages involved in the computation:

$$p_{m_k^{(i)}} = \Psi_i(p_{m_{k-1}^{(j_1)}, \ldots, m_{k-1}^{(j_{d_i})}}) \,. \tag{4.2}$$

In general, the function $\Psi_i(\cdot)$ depends on the index $i$ of the considered output message and can rarely be computed in closed form. Nevertheless, numerical approaches can be followed, for example by sampling the pdf and representing the messages with discrete RVs.

The discretization of the messages can be limited to the analysis purpose or, as practical scenarios call for low complexity implementations, could be the intrinsic way the receiver operates. In other words, the receiver may operate on discrete messages, e.g., the representation of the messages could be limited to 4 bits, in which case the messages belong to a 16 elements set. In a quantized scenario with $M$-level messages, the pdfs become probability mass functions (pmfs) which can be represented by $M$-element real vectors. The resulting pmf evolution function $\Psi$ is

$$\Psi : \mathbb{R}^{M^{d_i}} \mapsto \mathbb{R}^M \,.$$

Observing (4.1) and (4.2), one can immediately notice two issues:

i. if at the $k$-th iteration the joint pdf of the messages $m_{k-1}^{(j_1)}, \ldots, m_{k-1}^{(j_{d_i})}$ is needed, the computation of the marginal pdf $p_{m_k^{(i)}}$ only does not allow to proceed iterating the algorithm;

ii. even if the pdf $p_{m_k^{(i)}}$ is available for every $k$, how can one understand if the decoding algorithm is converging, i.e., if the BER is approaching 0?

A solution for the first issue is to assume conditional independence of the messages $m_{k-1}^{(j_1)}, \ldots, m_{k-1}^{(j_{d_i})}$. In this case, (4.2) becomes

$$p_{m_k^{(i)}} = \Psi_i(p_{m_{k-1}^{(j_1)}}, \ldots, p_{m_{k-1}^{(j_{d_i})}}) \tag{4.3}$$

which entails significant simplification of the problem, as can be seen in the discrete message case with $M$-level messages. In this case, the pmf evolution function $\Psi$ is

$$\Psi : \mathbb{R}^{Md_i} \mapsto \mathbb{R}^M .$$

Although, in general, this would be an approximation, this assumption holds exactly in the case of LDPC codes if the iteration number $k$ is smaller than the girth of the graph.

Consider now the second issue, i.e., how to track the actual convergence of the decoding system based on the evolution of the pdfs $p_{m_k^{(i)}}$. At the final iteration, the processing block outputs a vector $\hat{\boldsymbol{a}}$, which is an estimate of the transmitted data sequence, computed as a function of the last message set $\boldsymbol{m}_\ell$ and of the received observable sequence $\boldsymbol{r}$. Therefore, the output value of a symbol $\hat{a}_l$ is a function $g_l(\boldsymbol{m}, \boldsymbol{r})$. A corresponding pmf $p_{\hat{a}_l}$ can be computed as a function of the pdf of $\boldsymbol{m}$. Once obtained the pmf of $\hat{a}_l$ and given that the analysis is done assuming a particular transmitted sequence, it is easy to compute the corresponding probability of error.

Another common assumption done in a density evolution analysis of LDPC codes is that of considering a common distribution equal for all the input messages. In other words, this corresponds to assuming that the input pdfs are all equal:

$$p_{m_{k-1}^{(j_1)}} = \ldots = p_{m_{k-1}^{(j_{d_i})}} \triangleq p_{m_{k-1}} .$$

Since, at each step, $\{p_{m_k^{(i)}}\}$ are computed and generally vary for different values of $i$, this requires *an additional step* after the computation of each relevant output pdf $p_{m_k^{(i)}}$. In this additional step, usually, averaging of all output pdfs is performed for computing the pdf $p_{m_k}$ used as input pdf at the (next) iteration $k+1$:

$$p_{m_k} = \frac{1}{L} \sum_{i=1}^{L} p_{m_k^{(i)}}$$

where $L$ is the number of messages, i.e., the length of $\boldsymbol{m}_k$.

The assumption of a unique distribution for all input messages holds exactly in the case of *regular* LDPC codes [58], assuming that the all-zero sequence has been transmitted (present in all LDPC codebooks). In general, in a communication system, the performance depends on the particular transmitted codeword. However, the sum-product (SP) algorithm has a *symmetry property* that allows to state that the statistical description of the messages does not depend on the particular codeword. Therefore, the use of density

evolution leads, in this case, to exact results. For more details on density evolution techniques, which in general can be applied to a wide variety of message passing decoders, we refer the interested reader to [58].

We point out that the main disadvantage of this technique is that analytical derivation of the message pdf evolution function $\Psi(\cdot)$ is seldom feasible (with some important exceptions—see, for example, [13]). On the other hand, a statistical evaluation of the pdf evolution would need intensive Monte Carlo simulations, thus limiting the computational efficiency of this analysis technique.

## 4.4 EXIT Charts

Whenever its underlying assumptions hold, density evolution leads to a complete statistical characterization of the decoder, although it requires the capability of efficiently computing the evolution of an entire pdf (or probability mass function, pmf, if the message set is finite). Another possible solution is to track the evolution of some statistical function of the message set. This could lead to great simplifications. For example, in [59], the authors use a real valued function of the pdf of message sets, which is defined as an equivalent signal-to-noise ratio (SNR), as a means for characterizing the input-output relation of a processing block (the forward-backward, FB, algorithm for a particular convolutional code, in this case). The input SNR/output SNR relation of each component block of the iterative receiver is then used for predicting the convergence behavior of the receiver.

A statistical parameter, function of the message distribution, which allows to obtain good accuracy through this analysis method is the average mutual information (MI) between the generic transmitted codeword bit and the generic message in the decoder referring to that particular bit. In a SP decoder or, more generally, in a message passing decoder for LDPC codes, a message is said to *refer to a codeword bit* if it is originated from or is directed towards the variable node corresponding to that particular bit.

There is a number of possible techniques which can be used to compute the average MI of the messages. If every message $m$ in the set has the same known distribution $p(m|a)$, where $a$ is the corresponding bit, then the MI can be computed as usual:

$$I = \sum_{a} \int p(m|a)P(a) \log \frac{p(m|a)}{p(m)} \, \mathrm{d}m. \tag{4.4}$$

However, reality is usually more complicated: (i) the distributions of the mes-

sages are not equal and (ii) the message pdf is not known. In the first case, a possible definition of the MI is the following:

$$I = \frac{1}{L}\sum_{i=1}^{L} I_i \tag{4.5}$$

where $I_i$ is the MI between the $i$-th message and the corresponding bit, and $L$ is the total number of messages.

In the second case, i.e., whenever the distribution is difficult to compute, several approximate techniques may be used. In particular, in some circumstances, it may be convenient to use a Monte Carlo approach to obtain an estimate of the MI. This can be done by setting up a simulator and generating sets of sample messages which are quantized and used to obtain a histogram approximation of the pdf $p(m|a)$. The MI can therefore be computed based on the histogram approximation and using (4.4).

The MI has a practical meaning: it quantifies, in terms of bits, how much information about a particular codeword bit a given message carries. In practice, whenever the MI is equal to 1, the bit is reliably recovered. This can be easily shown: in fact, if $A$ is a uniformly distributed binary random variable, $H(A) = 1$. Assuming $I(A;M) = 1$, a functional relation exists between $M$ and $A$. In fact:

$$\begin{aligned} I(A;M) &= H(A) - H(A|M) \\ &\Downarrow \\ 1 &= 1 - H(A|M) \\ &\Downarrow \\ H(A|M) &= 0 \end{aligned}$$

which implies that, given $M$, $A$ is known, i.e., it is a function of $M$.

### 4.4.1 EXIT Curves and EXIT Charts

At this point, one could observe that although it is possible to track the evolution of the MI of the message sets during the iteration process, the MI is still a function of the underlying pdf of the messages and, therefore, little advantage may be obtained in this way. However, empirical observation shows that if a message set characterized by an MI $I'$ is used as input to a given processing block, the MI $I''$ of the output message set depends almost only on $I'$ and has little dependence on the particular pdf of the input messages. Each processing block can therefore be characterized by the relation between

the input message set MI and the output message set MI. Since, usually, only extrinsic information is exchanged between processing blocks, this the plot of this relation is referred to as *extrinsinc information transfer (EXIT) curve*. EXIT curves are used to study convergence of recursive (iterative) detection/decoding algorithms by means of graphs usually referred to as *EXIT charts*. EXIT chart-based analyses allow to predict the system performance with a significantly lower computational burden with respect to the use of density evolution or standard computer simulations employed to evaluate the BER performance of iterative decoders [60, 61].

The assumption of independence on the particular message set pdf is an approximation, although it turns out to be an effective one. A more rigorous statement is that, given an input MI, the output MI is bounded within a range of possible values whose extremes are functions of the input MI. This range is, in practical applications, reasonably small. A thorough treatment of this topic, usually referred to as *information combining*, can be found in [62–64].

A remark is worthwhile at this point. We said that the processing block modifies the MI. In particular, we wish the MI to increase, at each iteration, in order to approach the value 1 as closely as possible. One can observe, however, that by the data processing inequality [1], it is not possible to increase the MI by means of data processing. In fact, the MI we refer to is not the MI between the codeword bits and the message set, i.e., the MI between two vectors. The MI used in EXIT charts is the MI between a message and its corresponding codeword bit, averaged over all messages in the message set. This means that the statistical dependence between different messages is purposely neglected. The processing block can therefore exploit this dependence in order to increase the EXIT chart MI.

### 4.4.2 SISO Detectors and EXIT Charts

A processing block computing bit reliabilities based on (i) a set of constraints, (ii) an optional set of observations from the channel and (iii) some input *a priori* information, is usually referred to as soft-input soft-output (SISO) detector or SISO module [65–68]. In the following chapters, we will use SISO modules in systems employing differentially-encoded phase shift keying (DE-PSK) and DE quadrature amplitude modulation (DE-QAM) transmission over an AWGN channel, DE-PSK with noncoherent detection, and PSK transmission through a channel affected by ISI.

Since we will focus on binary coding techniques, we assume that the reliabilities at the output of SISO blocks are referred to binary symbols. This is not always the case, since algorithms like the FB algorithm in the general case

outputs reliabilities referring to $M$-ary symbols, where $M$ is the cardinality of the transmitted information symbol set. However, it is possible to transform $M$-ary reliabilities into a set of binary reliabilities and vice versa. The two operations are usually non-invertible, thus implying a possible information loss due to the conversion.

As an example, consider the conversion of the probabilities of an $M$-ary symbol $a$ taking values in the set $\{0, \ldots, M-1\}$ into bit reliabilities. Assume that $M = 2^n$, therefore the symbol $a$ can be represented by $n$ bits $b_0 \ldots b_{n-1}$. Given the probabilities $P\{a = 0\}, \ldots, P\{a = M-1\}$ we compute

$$P\{b_i = 0\} = \sum_{j:b_i=0} P\{a = j\}$$

where $j : b_i = 0$ is the set of all integers $j \in \{0, \ldots, M-1\}$ such that the $i$-th bit of their binary representation is equal to 0.

The conversion from bit reliabilities to symbol reliabilities can be done as follows, assuming all the bits are independent (which is usually an approximation):

$$P\{a = j\} = \prod_{i=0}^{n-1} P\{b_i = \omega_i(j)\}$$

where $\omega_i(j)$ denotes the $i$-th bit in the binary representation of $j$.

As already mentioned, EXIT curves are based on the computation of the MI between each binary symbol and its reliability. Due to the presence of binary symbols, this MI takes on a value between zero and one.

An EXIT curve for a SISO block $\mathbf{S}$ is a function $I_{\mathbf{S}}(I)$ which quantifies the average relationship between the MI of the reliabilities at the input of the block (i.e., the variable $I$) and the MI of the *a posteriori* reliabilities at the output of the block (i.e., $I_{\mathbf{S}}$)—recall that the MI is computed with respect to the transmitted information sequence [60, 61].

**Example 4.2** *Using a Monte Carlo simulation-based method for computing the EXIT curve of a SISO block*

As an example, consider an AWGN inter-symbol interference (ISI) channel with binary phase shift keying (BPSK) at its input. The data is transmitted in blocks of $N$ bits $(a_1, \ldots, a_N)$. At the receiver a SISO block:

- observes the output $\boldsymbol{r}$ of the channel;
- accepts a vector $(m_1^{(\text{in})}, \ldots, m_N^{(\text{in})})$ of $N$ *a priori* probabilities for the transmitted bits, i.e.,

$$m_i^{(\text{in})} = P\{a_i = 1\}\,;$$

- computes a vector $(m_1^{(\text{out})}, \ldots, m_N^{(\text{out})})$ of the a posteriori probabilities of the bits using the FB algorithm (2.16), i.e.,

$$m_i^{(\text{out})} = P\{a_i = 1 | \boldsymbol{r}\}\,.$$

Although it has no implication in this particular example, assume also that the output messages represent the extrinsic information, as described in Section 2.4.3. This is a common and important assumption in an iterative detection scheme [46].

Assume that we want evaluate the EXIT curve of the considered SISO block using the previously introduced Monte Carlo simulation-based method.

The SISO block has, as a matter of fact, two (vector) inputs: $\boldsymbol{r}$ and $(m_1^{(\text{in})}, \ldots, m_N^{(\text{in})})$. In an iterative decoding process, however, $\boldsymbol{r}$ is fixed, i.e., it does not change during the iterations. The EXIT curve must characterize the MI between the generic transmitted bit $a_i$ and the corresponding output message $m_i^{(\text{out})}$ *as a function of* the MI between the generic transmitted bit $a_i$ and the corresponding input message $m_i^{(\text{in})}$. To this end:

1. fix the SNR at the receiver;
2. generate the bit sequence $(a_1, \ldots, a_N)$ using a proper PRNG;
3. simulate the transmission of the bit sequence through the channel, obtaining the vector of observables $\boldsymbol{r}$;
4. generate a vector of messages $(m_1^{(\text{in})}, \ldots, m_N^{(\text{in})})$ characterized by an MI equal to $I$, as will be shortly discussed;
5. run the FB algorithm obtaining the output APPs $(m_1^{(\text{out})}, \ldots, m_N^{(\text{out})})$.

Assuming we have a method for generating the input vector $(m_1^{(\text{in})}, \ldots, m_N^{(\text{in})})$, the analysis, i.e., the output MI computation, proceeds as follows. Consider the pairs $\{(a_i, m_i^{(\text{out})})\}$ as samples of pairs of RVs distributed according to a pdf $p(a, m)$. Given the sample pair sequence, we wish to estimate the MI between $a$ and $m$, which represents the output MI. There are several methods for evaluating the MI from a sample sequence. A very simple one is deriving a histogram to estimate $p(a, m)$ and then computing the MI of the corresponding joint discrete RVs. Let us follow this simple method. Note that, in the considered scenario, the messages belong to $[0, 1) \subset \mathbb{R}$ and, therefore, to derive a histogram one needs to define a quantization rule. To this end, divide the interval $[0, 1)$ into $L$ bins $B_1, \ldots, B_L$, each of width $1/L$. At this point, it is

possible to associate the vector $(m_1^{(\text{out})}, \ldots, m_N^{(\text{out})})$ with a quantized vector $(\tilde{m}_1^{(\text{out})}, \ldots, \tilde{m}_N^{(\text{out})})$, where

$$\begin{aligned} \tilde{m}_i^{(\text{out})} &= \min\{j : m_i^{(\text{out})} \in B_j\} \\ &= \left\lfloor m_i^{(\text{out})} L \right\rfloor . \end{aligned}$$

The correct choice of the number of bins $L$ is important and should be chosen so that $1 \ll L \ll N$. The two extremal choices, $L = 1$ or $L = N$, will result in a MI equal to 0 and 1, respectively. Clearly the quantization operation would not have been necessary, had we analyzed a SISO block operating with quantized messages. The histogram approximation can therefore be obtained based on the sequence $(a_1, \ldots, a_N)$ and the quantized message sequence and is represented by the following joint pmf:

$$\tilde{p}(a, \tilde{m}) = \frac{1}{N} \sum_{i=1}^{N} 1(a_i = a \wedge \tilde{m}_i^{(\text{out})} = \tilde{m})$$

where $1(\cdot)$ is the indicator function previously introduced and $\wedge$ denotes the logical AND.

Given the estimate pmf of the quantized messages, we can compute the MI between a transmitted bit $A$ and the corresponding message $M$ using the following approximation:

$$I(A; M) \simeq \sum_{a} \sum_{\tilde{m}} \tilde{p}(a, \tilde{m}) \log_2 \frac{\tilde{p}(a, \tilde{m})}{\tilde{p}(a)\tilde{p}(\tilde{m})}$$

where

$$\tilde{p}(\tilde{m}) = \sum_{a} \tilde{p}(a, \tilde{m})$$

and

$$\tilde{p}(a) = \sum_{\tilde{m}} \tilde{p}(a, \tilde{m}) .$$

By changing the MI $I$ characterizing the input set, and by re-performing all the above described steps, one can obtain a new set of output messages and compute the new output MI $I_S(I) = I(A; M)$, thus obtaining all desired points of the EXIT curve $I_S(I)$.

All the above considerations assume we can generate a vector $(m_1^{(\text{in})}, \ldots, m_N^{(\text{in})})$ of input messages characterized by an MI $I(A; M^{(\text{in})})$ equal to $I$. A possible solution is starting with an input vector of probabilities all equal to 1/2, i.e.,

declaring to the SISO block that there is no *a priori* information on the transmitted bits. This corresponds to start with $I = 0$. We can compute the output message vector and characterize its MI $I_S(0)$. At this point, the output message vector is a vector of probabilities for which we know the MI and, in principle, could be used as input vector for the estimation of $I_S(I_S(0))$. This can be recursively combined to obtain several point of the EXIT curve. However, this approach could lead to inaccurate results and requires to keep the same transmitted bit sequence since all message set will refer to that particular sequence. A common approach to generate an *a priori* probability message sequence for a bit sequence $(a_1, \ldots, a_N)$ is as follows.

Considering a BPSK transmission over an AWGN channel, fix the channel noise variance $\sigma^2$ so that the MI between the input and the output of the channel is equal to $I$. This can be done numerically by inverting the MI expression given in Example 1.1. Transmit the bit sequence $(a_1, \ldots, a_N)$ through the obtained channel, i.e., for each bit $a_i$ apply the BPSK mapping rule, obtaining a transmitted symbol $c_i$, and add an AWGN noise sample characterized by variance $\sigma^2$, obtaining an output observable $y_i$. Now, compute the *a posteriori* probability of the bit:

$$m_i = \frac{e^{-\frac{(y_i+1)^2}{2\sigma^2}}}{e^{-\frac{(y_i+1)^2}{2\sigma^2}} + e^{-\frac{(y_i-1)^2}{2\sigma^2}}} . \tag{4.6}$$

Since $m_i$ is an invertible function of $y_i$, by the data processing inequality,

$$I(A; M) = I(A; Y) = I .$$

This implies that the vector $(m_1, \ldots, m_N)$ is a vector of messages, where each element represents the probability that the corresponding transmitted bit is equal to 1 and whose MI (i.e., the MI between the message and the transmitted bit RVs) is equal to $I$.

Note that there are infinite methods of generating messages characterized by a MI equal to $I$, and each method is characterized by a conditional distribution of the messages. The above described method is particularly interesting since the generated message sequence, in the log-likelihood domain, is conditionally Gaussian. This is an appealing property, since there are several useful SISO block output messages that are characterized by a Gaussian distribution in the log-likelihood domain. In other words, a message set generated with the above described method will exhibit statistical properties similar to those of a real SISO block.

## 4.5 EXIT Charts for LDPC Codes

The following example describes a method to compute the EXIT curves associated with the VND and the CND, i.e., the component blocks of a standard LDPC decoder.

**Example 4.3** *EXIT curves for the belief propagation LDPC decoder: Gaussian approximation*

Consider the variable and check node algorithm (3.9) and (3.10), involving messages in the LLR domain . Assume to approximate the distribution of the messages with a Gaussian distribution and assume also that all the input messages have equal distribution (which, as stated in the previous section, is a common assumption in density evolution analysis). In [13], it is shown that the distribution $p(m)$ of a message $m$ in an LDPC belief propagation decoder in the LLR domain must fulfill the following symmetry condition:

$$p(m) = e^m p(-m)$$

which, for a Gaussian distribution, implies that

$$\mathrm{Var}\{m\} = 2\mathrm{E}\{m\}\,.$$

Therefore, tracking the variance is sufficient to completely describe the evolution of the distribution.

Assuming, without loss of generality, the transmission of an all-0 sequence, given a Gaussian LLR message set fulfilling the above symmetry condition, the MI between the generic message and the corresponding codeword bit is [69]

$$J(\sigma) \triangleq \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\sigma^2/2)^2}{2\sigma^2}} \log_2 \frac{2}{1+e^{-x}}\,\mathrm{d}x\,. \tag{4.7}$$

where $\sigma$ denotes the standard deviation of the message set.

Consider now the operation (3.9) performed during iterative decoding by the variable node and operating in the log likelihood domain. Assuming that the messages at the input of the variable node are independent, the variance of the output message will be equal to the sum of the variances of the input messages. As a consequence, the input-output relation of the MI in a degree $d_v$ variable node, under the Gaussianity assumption, will be:

$$I_{\mathrm{out}} = J\Big(\sqrt{(d_v-1)(J^{-1}(I_{\mathrm{in}}))^2 + (J^{-1}(I_0))^2}\Big) \tag{4.8}$$

where $I_{\mathrm{out}}$ and $I_{\mathrm{in}}$ denote the MI between the transmitted codeword bit and a corresponding output and input message, respectively and

1. $$J^{-1}(\cdot)$$
   is the inverse of the $J(\cdot)$ function

2. $$J^{-1}(I_{\text{in}})$$
   is the standard deviation $\sigma_{\text{in}}$ associated with the input message set

3. $$(d_v - 1)(J^{-1}(I_{\text{in}}))^2$$
   is the sum of the variances of the input messages

4. $I_0$ is the MI between the *external observation* (leading in (3.9) to the LLR $m_0$) and the corresponding codeword bit

5. $$(J^{-1}(I_0))^2$$
   is the variance of the message associated with the *external observation*

6. and, finally,
   $$\left(\sqrt{(d_v - 1)(J^{-1}(I_{\text{in}}))^2 + (J^{-1}(I_0))^2}\right)$$
   is the standard deviation of the message at the output of the variable node.

If the LDPC code is irregular, one can obtain an average MI associated to the generic message from the VND and input to the CND according to (4.5), which leads to

$$I_{\text{out}}^{\text{VND}} = \sum_i \lambda_i J\left(\sqrt{(i-1)(J^{-1}(I_{\text{in}}))^2 + (J^{-1}(I_0))^2}\right) \tag{4.9}$$

where $\{\lambda_i\}$ are the variable node degree distribution coefficients

An approximate formula for the input/output relation for a degree-$d_c$ check node, based on a property of the BEC, is given by

$$I_{\text{out}} = 1 - J\left(\sqrt{d_c - 1}\, J^{-1}(1 - I_{\text{in}})\right). \tag{4.10}$$

For a detailed overview on how to derive the approximate formula (4.10) and the exact one, we refer the interested reader to [70, 71].

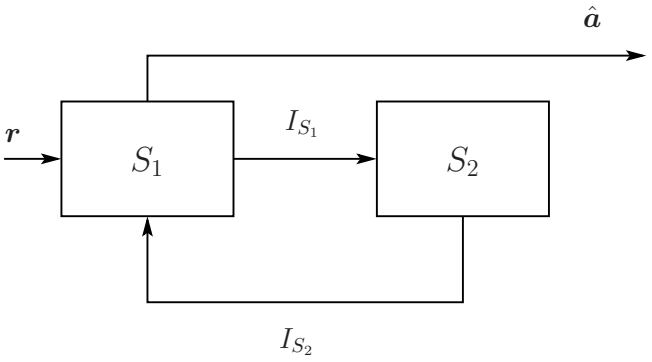


Figure 4.4: Schematic diagram of an iterative receiver comprising two SISO modules.

The average MI at the output of the CND, computed according to (4.5), is as follows:

$$I_{\mathbf{B}} = 1 - \sum_j \rho_j J\Big(\sqrt{j-1}\, J^{-1}(1 - I_{\mathbf{A}})\Big) \tag{4.11}$$

where $\{\rho_j\}$ are the check node degree distribution coefficients.

The MI of the message, computed according to (3.11), at the output of a degree-$d_v$ variable node at the last iteration is

$$I_{\text{out}} = J\Big(\sqrt{(d_v)(J^{-1}(I_{\text{in}}))^2 + (J^{-1}(I_0))^2}\Big) \tag{4.12}$$

and the corresponding average MI is

$$I_{\text{out}}^{\text{VND}} = \sum_i \lambda_i J\Big(\sqrt{(i)(J^{-1}(I_{\text{in}}))^2 + (J^{-1}(I_0))^2}\Big)\ . \tag{4.13}$$

In the following chapters, the receiver will be divided into two distinct processing blocks. This allows to simplify the analysis by decomposing the MI input-output relation into two simpler functions. A generic example of this scheme is shown in Figure 4.4, where two SISO modules $S_1$ and $S_2$ are connected in a turbo-like configuration. In Figure 4.5, the EXIT curves of these two hypothetical blocks $S_1$ and $S_2$ are shown. In the graph, the horizontal axis refers to the output MI of SISO module $S_2$ and the vertical axis refers to the output MI of SISO module $S_1$, i.e., the inverse $I_{S_2}^{-1}(I)$ of the $S_2$ EXIT curve is actually plotted. This representation of a pair of EXIT curves is referred to as *EXIT chart*, and is useful to investigate the decoding process as a recursive

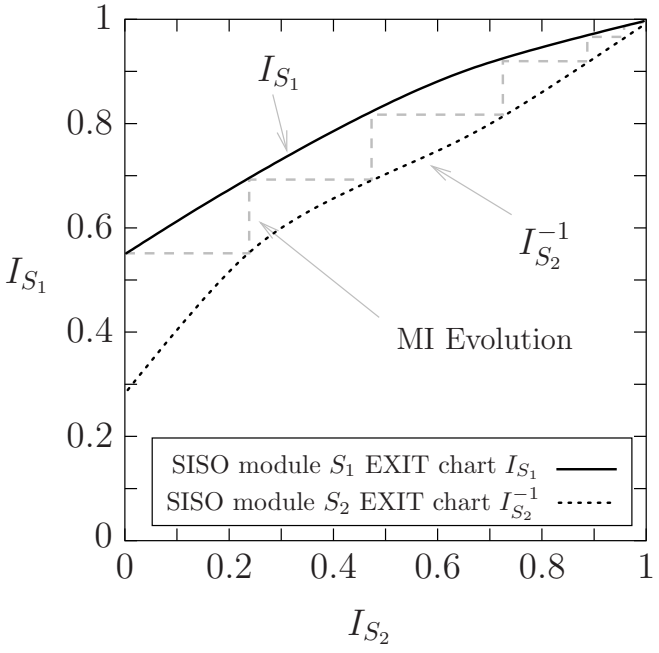


Figure 4.5: Example of EXIT chart: two SISO modules $S_1$ and $S_2$ iteratively exchange messages; the evolution trajectory of the MI is also shown.

update of the MI. A trajectory representing the evolution of the MI at the output of the SISO modules $S_1$ and $S_2$ in the EXIT chart is also shown. If the MI becomes equal to 1, the decoding process is said to *converge*, in the sense that a low BER can be expected.

In Chapter 5, the relation between the MI and the BER will be investigated and used for LDPC code design purposes.

## 4.6 Concluding Remarks

In this chapter, the main analysis tools for iterative receivers have been discussed. In particular, EXIT chart-based analyses will play an important role in the rest of the book, since it provides a simplified, yet accurate, convergence analysis tool which is well suited for LDPC code design for coded modulations.