Advanced Coding

PC Sessions

Rotondo Massimiliano	199699
Ferro Demetrio	207872
Minetto Alex	211419

Contents

PC Session 1: BER Counter **1**
Purpose . . . 1
Problem Statement . . . 1
Available software . . . 1
System Description . . . 2
Results Discussion . . . 3
BER Meter Class Implementation . . . 5

PC Session 2: Limits of Coding **7**
Purpose . . . 7
Problem Statement . . . 7
Available (new) software . . . 7
System Description . . . 7
Constrained Signal Set Analysis . . . 8
Finite Block Length Analysis . . . 10

PC Session 3: SISO Decoder **15**
Purpose . . . 15
Problem Statement . . . 15
Available (new) software . . . 15
System Description . . . 16
SISO Implementation . . . 18
Results Discussion . . . 20

PC Session 4: PCCC Decoder **23**
Purpose . . . 23
Problem Statement . . . 23
Available (new) software . . . 23
System Description . . . 23
PCCC Decoder Implementation . . . 25
Results Discussion . . . 25

PC Session 5: LDPC Decoder **31**
Purpose . . . 31
Problem Statement . . . 31
Available (new) software . . . 31
System Description . . . 31
LDPC Implementation . . . 32
Results Discussion . . . 35

PC Session 1: BER Counter

Purpose

1. Learn how to use Microsoft Visual C++ development environment
2. Understand the structure of a simulation program in C++ for a communication system
    a. The blocks of the communication system
    b. The signals
    c. Input parameters
    d. The simulation loop and simulation interval
    e. Output of measures on system
3. Learn how to implement a simple class for the simulation of a communication block
    a. Public and private members
    b. Common methods
4. Implement a class "**`BER_M`**" that emulates the block computing the Bit Error Rate by comparing two binary streams
    a. Check its functionality by substituting the implemented class to the class already available in TOPCOM and verifying that it provides the same results on the system considered in the first PC session
5. Writing a report
    a. Description of the system
    b. Description of the purpose of the simulation and measures performed over the system
    c. Presentation of the results
    d. Comments on the results

Problem Statement

Computation of the bit error probability as a function of the parameter Eb/N0 for systems using QAM modulations.

Available software (C++ classes)

1. The source of random bits (**`PN_Source`**)
2. The Modulator, mapping bits to QAM constellation points (**`Modulator`**)
3. The AWGN channel, adding Gaussian noise to the constellation points (**`AWGN_Channel`**)

4. The demodulator, performing the detection based on the minimum Euclidean distance criterion and returning the corresponding bits (**`Demodulator`**)
5. The BER (Bit Error Rate) Meter , comparing transmitted and decoded bit and computing the ratio between erroneous bits and total transmitted bits (**`BER_meter`**)
6. The main program (**`test_simple.cpp`**)

System Description

The aim of the Laboratory session is to understand the construction of a Simulator in C++ of a communication system and compute its performances in terms of Bit Error Rate for different QAM modulation schemes. The block diagram of the simulated channel is shown in Figure 1.

Each block is implemented in the simulator defining a proper class in C++.

The source produces a sequence $\bar{u}$ of random bits. The modulator maps sets of bits into QAM constellation points according to the size of constellation adopted.

The sequence $\bar{x}$ is transmitted over the AWGN channel where a sample of Gaussian Noise is added to each symbol. The received sequence $\bar{y}$ reaches the Demodulator that produces the bits sequence $\hat{\bar{u}}$ according to the minimum distance criterion from nominal points of the modulation scheme.

The Block BER Meter compares the received sequence $\hat{\bar{u}}$ with the sent sequence $\bar{u}$ and estimates the Bit Error Probability computing the ratio between the number of wrong bits and the total number of transmitted bits.

The parameters set by the user from an input file are the maximum number of frames f_{max}, the range of $\frac{E_b}{N_0}$ values, the modulation efficiency m, i.e. the number of bits carried by each QAM symbol.

Each frame is made by 20 modulated symbols, so that the total number of bits is $20 \cdot f_{max} \cdot m$.

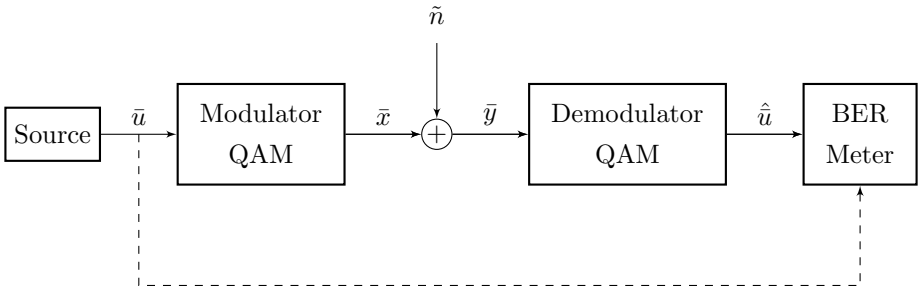


Figure 1: System Block Diagram

The time evolution of the simulation is frame by frame, implemented with a loop in which:

- It generates $20 \cdot m$ bits;
- It takes groups of m bits to obtain 20 symbols $\bar{x}$ with unitary average energy;
- It adds to each symbol a Gaussian sample $\tilde{n}$ with zero mean and variance $\frac{N_0}{2}$;
- It demodulates each received symbol $\bar{y}$ according to the Voronoi regions of the selected modulation scheme;
- It counts the number of error bits comparing the bit sequence $\bar{u}$ with the demodulated $\hat{\bar{u}}$;
- In order to consider the BER estimation reliable, we require to count at least 100 errors. When this condition is reached, a relative flag variable is set to true;

The simulator exits from the loop either if the reliability condition is verified, or if the maximum number of frames f_{max} is reached. The simulation is run for each value of $\frac{E_b}{N_0}$.

Results Discussion

Outputs of the simulator are the total number of generated bits, the total number of errors counted, the Bit Error Probability estimation computed as the ratio between the number of errors and the total number of bits.

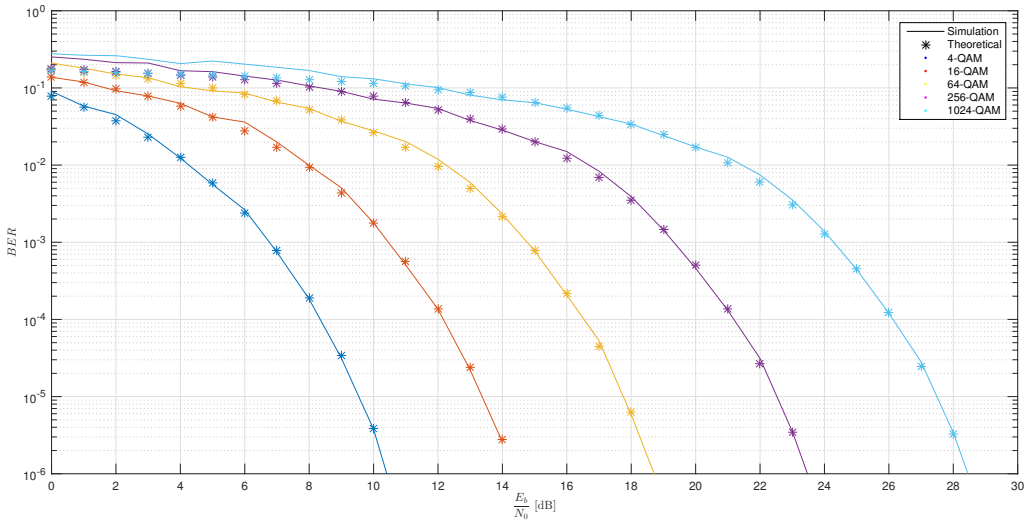


Figure 2: BER estimation for different QAM modulations.

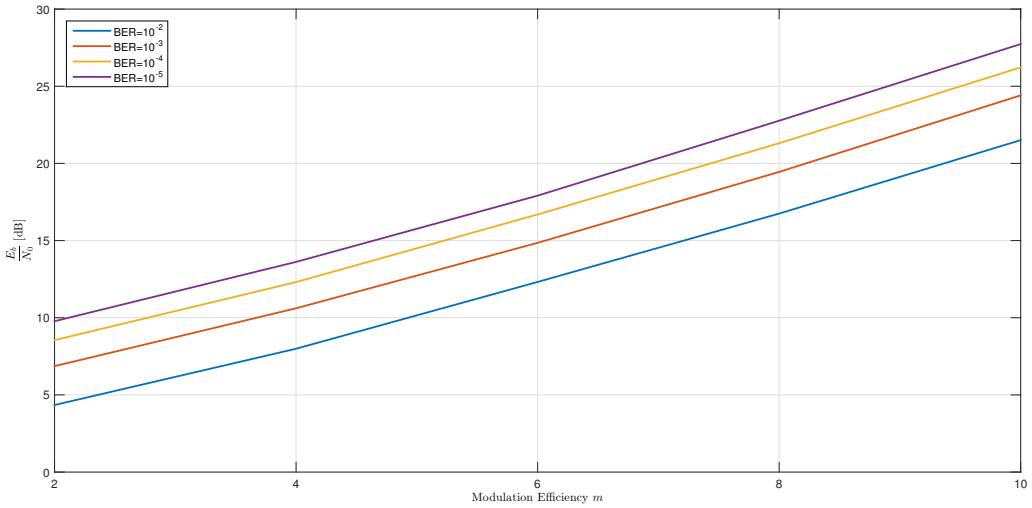


Figure 3: Required $\frac{E_b}{N_0}$ to reach a given BER Target varying with the Modulation Efficiency.

The simulation is executed with $f_{max} = 2 \cdot 10^6$, $\frac{E_b}{N_0} = [0, 30]$ dB and different values of $m = [2, 4, 6, 8, 10]$.
In Figure 2 we show the BER estimation curves obtained from the simulation compared with the theoretical ones, according to the generic expression for the QAM modulation:

$$P_b(e) = \frac{\sqrt{M} - 1}{\sqrt{M} \cdot \log_2 \sqrt{M}} \cdot \text{erfc}\left(\sqrt{\frac{3 \cdot \log_2 M}{2(M-1)} \frac{E_b}{N_0}}\right)$$

where $M = 2^m$ is the constellation cardinality.

In Figure 3 we report the required $\frac{E_b}{N_0}$ to reach a given BER target varying with the Modulation Efficiency.

We can notice that increasing the constellation cardinality, we need a larger value of E_b/N_0 to achieve a target Bit Error Rate.
This is due to the fact that increasing the constellation size and keeping equal to one the average energy, the distance between the nominal points reduces and so samples of noise with lower value are sufficient to carry out the transmitted symbols from the correct Voronoi region.
Anyway a modulation with a symbol that includes more bits, allows to increase the transmission rate.
So there is a trade off between Error Probability and rate requirements.

BER Meter Class Implementation

We implemented a simplified version of the available BER Meter class.
The header file contains the attributes and the methods of the class, all the attributes are declared private, so they are accessible only within the same class. Methods are instead public, so that they can be externally accessed.

The attributes are the following:

1. **`nerr`**, the number of errors;
2. **`ncountedbits`**, the number of counted bits;
3. **`minerr`**, the minimum number of counted errors to consider the BER measure reliable;
4. **`delay`**, the delay between the transmitted and the received sequence.

The methods are the following:

1. **`BER_M`** is the constructor of the class, it simply calls the method reset when an object of this class is created.

```
BER_M::BER_M()
{
  Reset();
}
```

2. **`~BER_M`** is the destructor of the class, it is called when the object is deleted to perform some useful instructions. In our case, we don't use it.

```
BER_M::~BER_M()
{
}
```

3. **`SetParameters`** sets the attributes `delay` and `minerr`.
   The delay is set to 0, since we assumed that the channel does not introduce delay.

```
void BER_M::SetParameters(const int delay, const int minerr)
{
  this->minerr = minerr;
  this->delay = 0;
  Reset();
}
```

4. **`Reset`** sets to 0 the attributes `nerr` and `ncountedbits`, it is called by the constructor of the class.

```
void BER_M::Reset()
{
  this->nerr = 0;
  this->ncountedbits = 0;
}
```

5. **`Run`** compares the transmitted and received sequences, updating `ncountedbits` and `nerr`.
   The error bits are checked by performing a modulo 2 sum operation.

```
void BER_M::Run(const int nbits, const int* stream1, const int* stream2)
{
  int i;
  for (i=0; i<nbits; i++)
  {
    ncountedbits++;
    nerr += stream1[i]^stream2[i];

  }
}
```

6. **`IsReliable`** verifies if `nerr` is greater than `minerr`, returning a boolean value that is used as exit condition from the loop of the simulation.
   The method is called at the end of each frame, after the `Run` method.

```
bool BER_M::IsReliable()
{
  if (nerr >= minerr) return true;
  else        return false;

}
```

7. **`Display`** saves in a file the ratio between `nerr` and `ncountedbits` that is the BER estimation, the value of `nerr` and `ncountedbits`.

```
void BER_M::Display(FILE* file)
{
  fprintf(file, "BER = %e\n", (double)nerr / ncountedbits);
  fprintf(file, "nerr = %d\n", nerr);
  fprintf(file, "ncounted bits = %d\n", ncountedbits);
}
```

8. **`Display_on_Line`** visualizes the same results shown by the method `Display` on `stdout` that has to be passed as argument.

```
void BER_M::Display_on_Line(FILE* file)
{
  fprintf(file, "%e\t", (double)nerr / ncountedbits);
  fprintf(file, "%d\t", nerr);
  fprintf(file, "%d\t", ncountedbits);
}
```

The header file **`BER_M.h`** is the following:

```
#pragma once
#include <stdio.h>

class BER_M
{
public:
  BER_M();
  ~BER_M();

  void SetParameters(const int delay, const int minerr);
  void Reset();
  void Run(const int nbits, const int* stream1, const int* stream2);
  bool IsReliable();
  void Display(FILE* file = stdout);
  void Display_on_Line(FILE* file = stdout);

private:
  int nerr;
  int ncountedbits;
  int minerr;
  int delay;

};
```

PC Session 2: Limits of Coding

Purpose

Computation of lower bounds to achievable frame error probability for several block sizes, codulator rates and 2 dimensional modulations

Problem Statement

1. Compute lower bounds on achievable performance (FER) for
   a. Information Block sizes 100, 1000, 10000, 100000, Infinity
   b. Modulations (2, 8, 16)-PSK and (4, 16)-QAM
   c. FER 10^{-1} : 10^{-8}.
2. Write a report with the obtained results and comments.

Available (new) software

Program for the computation of the capacity of 2D modulations and sphere packing bound.

System Description

The aim of the Laboratory session is to compute the limits of coding techniques in terms of E_b/N_0 required to achieve a given mutual information.

First of all, we perform an analysis for infinite length block size. We compute the loss introduced by the use of finite 2D constellation cardinality with respect to the theoretical unconstrained results obtained by Shannon.

Then, we proceed by evaluating the worsening of performances in terms of Frame Error Rate (FER) when the block size progressively reduces.

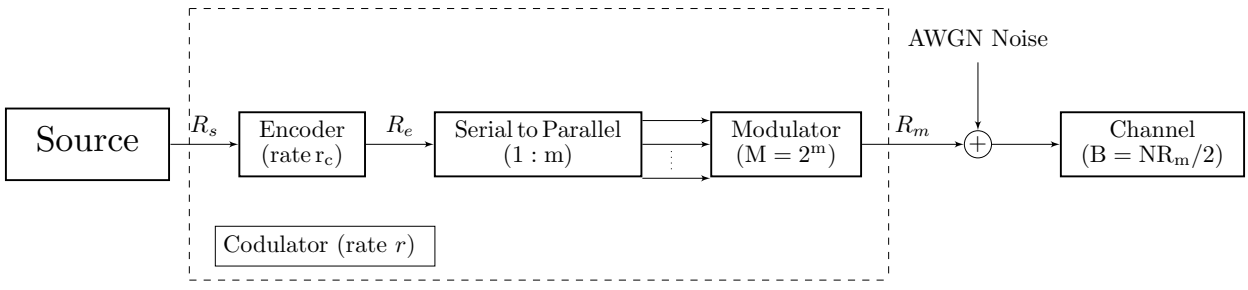


Figure 4: System Block Diagram

Let us consider the system shown in Figure 21 where:

- R_s [bits/s] is is the Information bit rate;
- R_e [bits/s] is the Coded bit rate;
- $r_c = \dfrac{R_s}{R_e}$ is the rate of the encoder;
- $R_m = \dfrac{R_e}{m}$ [Baud] is the symbol rate;
- $M = 2^m$ is the constellation cardinality;
- N is the dimensionality of the constellation.

The Encoder and the Modulator can be considered as a single system block called Codulator, which receives information bits in input at rate R_s and delivers channel symbols at rate R_m. So we may define the codulator rate as $r = \dfrac{R_s}{R_m} = \dfrac{mR_s}{R_e} = mr_c$.

Shannon's formula for Channel coding over AWGN channel is the following:

$$C = \frac{1}{2}\log_2\left(1 + \frac{2rE_b}{NN_0}\right) \tag{1}$$

Where $C = \sup_{p(x)} I(X,Y)$ is the channel capacity, $I(X,Y)$ is the mutual information of the channel.

The Shannon Theorem for AWGN channel imposes a limit on the rate, $0 \leq r \leq C$ of the communication, but ensures the existence of a code that achieves arbitrary low error probability for infinite block size.

By introducing the Bandwidth Efficiency: $\varepsilon_b = \dfrac{R_s}{B} = \dfrac{2R_s}{NR_m} = \dfrac{2r}{N}$

We can rewrite the previous formula as follows:

$$C = \frac{1}{2}\log_2\left(1 + \varepsilon_b\frac{E_b}{N_0}\right) \tag{2}$$

Since $0 \leq r \leq C$, by inverting the previous expression, the minimum E_b/N_0 required to achieve an infinitely reliable transmission for a given Bandwidth Efficiency is:

$$\frac{E_b}{N_0} = \frac{2^{\varepsilon_b} - 1}{\varepsilon_b} \tag{3}$$

Constrained Signal Set Analysis

The capacity has a closed-form expression for 2D constellation:

$$C = \log_2(M) - \frac{1}{2\pi}\iint_{-\infty}^{+\infty} d(u,v)e^{\frac{-(u^2+v^2)}{2}}dudv \tag{4}$$

where:

$$d(u,v) = \frac{1}{M}\sum_{j=1}^{M}\log_2\left(\sum_{i=1}^{M}\exp\left[-\frac{E_s}{N_0}\left[(a_j-a_i)^2+(b_j-b_i)^2\right] - \sqrt{\frac{2E_s}{N_0}}\left[u\cdot(a_j-a_i)+v\cdot(b_j-b_i)\right]\right]\right) \tag{5}$$

- $\frac{E_s}{N_0}$ is the average energy per symbol over the noise power spectral density;
- (a_i, a_j) and (b_i, b_j) are the normalized coefficients of the constellation points.

The expression (5) can be simplified if the constellation has a geometrical uniform distribution (such as M-PSK).

$$d(u,v) = \log_2 \left(\sum_{i=1}^{M} \exp \left[-\frac{E_s}{N_0} \left[(a_j - a_i)^2 + (b_j - b_i)^2 \right] - \sqrt{\frac{2E_s}{N_0}} \left[u \cdot (a_j - a_i) + v \cdot (b_j - b_i) \right] \right] \right) \tag{6}$$

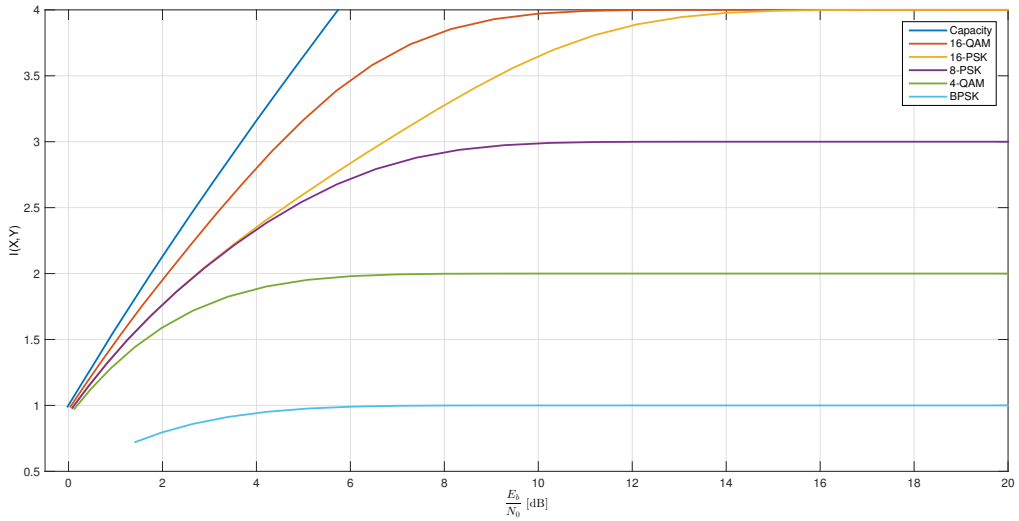


Figure 5: Capacity of 2D Constellations.

In Figure 5 we plot the Capacity as a function of the minimum E_b/N_0 required to achieve unlimited reliabity over an AWGN channel.

The plot is obtained as follows:

- Considering a range of $\frac{E_s}{N_0} = [0, 20]$ dB;
- Compute the Capacity by using the expressions (5), (6), with a numerical approximation for (4);
- The minimum E_b/N_0 for the given constellation is computed by the relationship: $\frac{E_b}{N_0} = \frac{1}{C} \cdot \frac{E_s}{N_0}$;
- The minimum E_b/N_0 for the unconstrained case is computed from (3);

As we can see from the obtained curves, using unconstrained modulation format, we can reach arbitrary high values of Mutual Information increasing E_b/N_0. By limiting the constellation's cardinality to M, instead, the values assumed by Mutual Information cannot exceed m, i.e. the number of bits carried by each symbol. The loss of finite constellations with respect to the Gaussian distribution derives from the Shaping Loss introduced by the modulation. We assume that all the points of the constellation are transmitted with same

probability $\frac{1}{M}$, differently from the optimal Gaussian distribution. The shaping loss of M-QAM with large M is about 1.53 dB for high values of E_b/N_0, while for low values ($E_b/N_0 \simeq 0$ dB) it is negligible. Infact, apart from BPSK case (corresponding to 1D constellation), all curves behave like Shannon's in that region. Using constellation shaping technique is possible to produce a Gaussian-like distribution over constellation in order to reduce the gap of 1.53 dB.
The reliability for M-QAM constellations is reached for lower values of E_b/N_0 with respect to M-PSK since the minimum distance between two adjacent points is larger.
Moreover, using a constellation with greater cardinality allows to save E_b/N_0 to reach a target rate.

Finite Block Length Analysis

Achieving an infinitely reliable communication is possible under the assumption of using blocks of infinite size if E_b/N_0 is greater than the value expressed by (3). In practice, this assumption can not be adopted for latency and memory requirements. In spite of this we introduce the Frame Error Rate (FER) as new parameter to design the code. It is defined as the ratio between the number of correct received block sequences and the number of total transmitted sequences. In order to understand properly the performances of a code, it is suitable to find a lower bound that ensures that it does not exist a code that can reach better performances. The sphere packing gives a lower bound to the FER for a code under the assumption of adopting an unconstrained modulation. The bound is in the form $P_w(e) \geq Q_n(\theta, A)$, where:

- $P_w(e)$ is the word error probability;
- $Q_n(\theta, A) \simeq \dfrac{\left|G \sin\theta \exp\left(-(A^2 - AG\cos\theta)/2\right)\right|^n}{\sqrt{n\pi(1+G^2)\sin\theta\left|AG\sin^2\theta - \cos\theta\right|}}$;
- $G = \dfrac{1}{2}A\cos\theta + \sqrt{A^2\cos^2\theta + 4}$;
- $A = \sqrt{2\dfrac{k}{n}\dfrac{E_b}{N_0}}$;

We can notice that A depends on the information block size k and on E_b/N_0.
A reasonable assumption is that the loss of the sphere packing bound at a given rate with respect to the unconstrained capacity can be applied also to the constrained capacity, leading to the relationship:

$$\left.\frac{E_b}{N_0}\right|_{\mathcal{M},P_w,k} \geq \left.\frac{E_b}{N_0}\right|_{\mathcal{M}} + \left(\left.\frac{E_b}{N_0}\right|_{\mathcal{U},P_w,k} - \left.\frac{E_b}{N_0}\right|_{\mathcal{U}}\right) \tag{7}$$

where:

- $\left.\frac{E_b}{N_0}\right|_{\mathcal{M},P_w,k}$, $\left.\frac{E_b}{N_0}\right|_{\mathcal{U},P_w,k}$ are the minimum required E_b/N_0 for a code of length k to achieve a word error probability P_w using respectively the modulation $\mathcal{M}$ and the unconstrained Gaussian modulation $\mathcal{U}$;
- $\left.\frac{E_b}{N_0}\right|_{\mathcal{M}}$, $\left.\frac{E_b}{N_0}\right|_{\mathcal{U}}$ are the minimum required E_b/N_0 for an infinite block length $k \to \infty$ code using respectively the modulation $\mathcal{M}$ and the unconstrained Gaussian modulation $\mathcal{U}$;

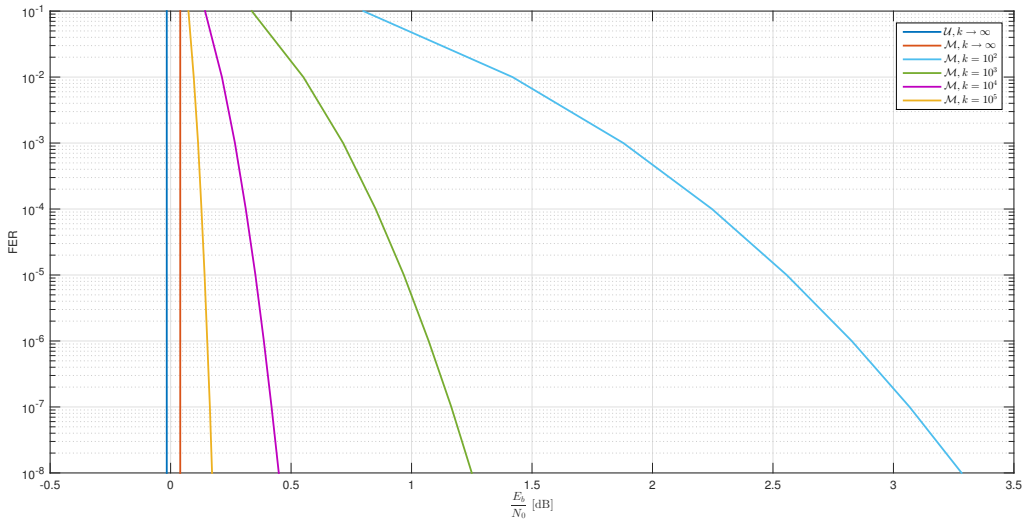


Figure 6: Lower Bounds of FER for 16-QAM with $r = 0.99$, $k = 100, 1000, 1000, 10000, k \to \infty$.

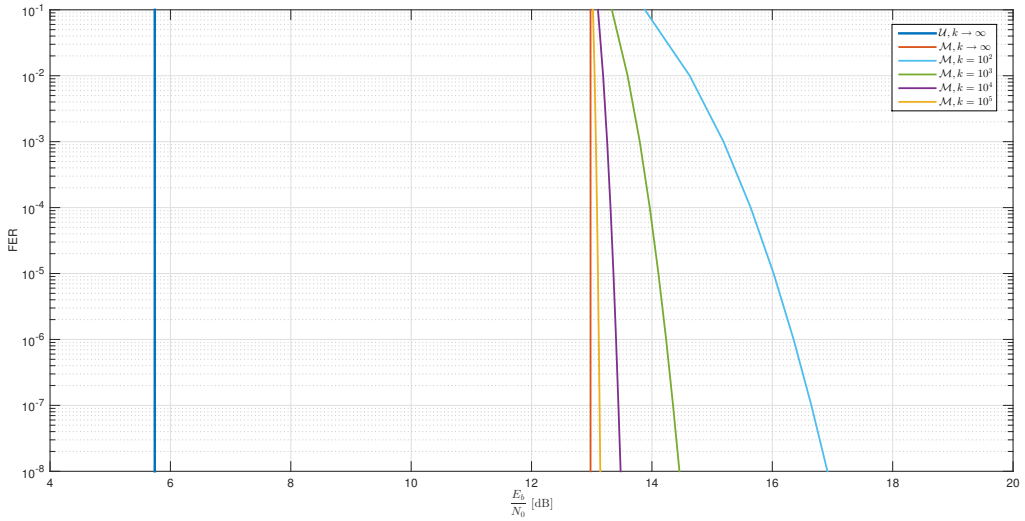


Figure 7: Lower Bounds of FER for 16-QAM with $r = 3.99$, $k = 100, 1000, 1000, 10000, k \to \infty$.

In Figure 6 and 7 we plot the lower bounds of the FER as function of E_b/N_0 using 16-QAM modulation, for different block size $k = 10, 100, 1000, 10000, 100000$ and for codulator rates 0.99 and 3.99 respectively.
As we expect the Infinite Block Length size curves have a vertical behaviour according to the Shannon Theorem. With respect to the Unconstrained Case, there are two types of loss: one introduced by the constrained modulation and the other one due to the block size.
The loss due to the modulation is strictly related to the codulator rate. The losses of E_b/N_0 are 0.056 dB for $r = 0.99$ and 7.24 dB for $r = 3.99$.
For low rate values, the loss is smaller since the 16-QAM rate curve behaves like the Capacity as we can see in Figure 5.
For $r = 3.99$ the curve is in a region where the slope is much lower and so, it requires an higher E_b/N_0 to

achieve that rate.

We can notice in Figure 6 and 7 that reducing the block size, we require larger values of E_b/N_0 to achieve the same FER value. Differently from the loss due the modulation, the loss due to the block size is almost independent from the codulator rate.

We can appreciate this behaviour in Figure 8 where we plot the loss of E_b/N_0, using 16-QAM modulation, required to achieve a $FER = 10^{-4}$ varying the block size respect to the Infinite block size case.

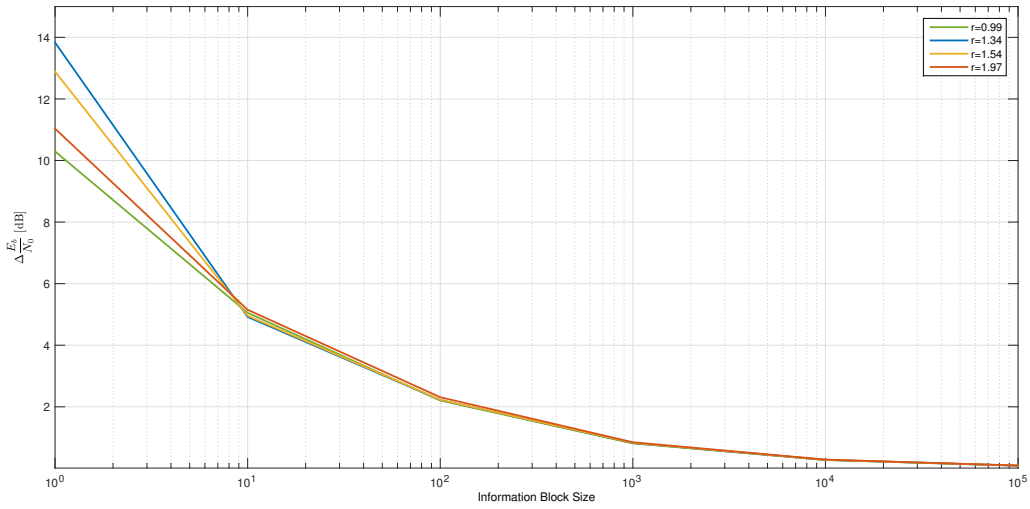


Figure 8: E_b/N_0 Loss of 16-QAM modulation for different block sizes and rates at $FER = 10^{-4}$.

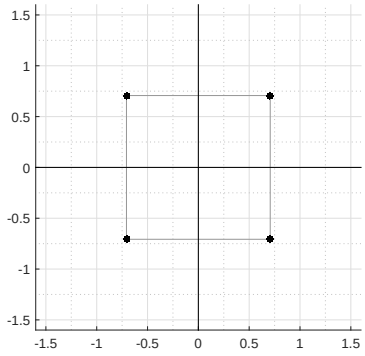
In the following plots, (Figure 9b, 10b, 11b, 12b) we report the lower bounds of FER using different modulation formats and block sizes for the same codulator rate $r \simeq 2$. The loss introduced by the Information Block Size is independent from the adopted modulation. The loss for each block size respect to the Infinite Block Size case is the same for each modulation.

From the capacity we need 1.76 dB to achieve a rate equal to 2.

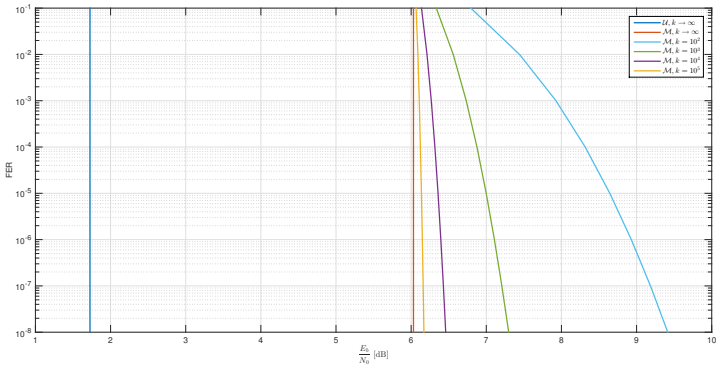
In Table 1 we report the E_b/N_0 loss respect the capacity values for different modulations and block size. We can notice that choosing a greater block length and constellation size reduces the loss. The losses of 16-PSK and 8-PSK are almost the same for $r = 2$. The 4-QAM has a greater loss with respect the other modulations.

	$k = 100$	$k = 10^3$	$k = 10^4$	$k = 10^5$	$k \to \infty$
4-QAM	6.59 dB	5.15 dB	4.59 dB	4.40 dB	4.31 dB
8-PSK	3.34 dB	1.90 dB	1.34 dB	1.15 dB	1.05 dB
16-PSK	3.32 dB	1.90 dB	1.33 dB	1.14 dB	1.04 dB
16-QAM	2.64 dB	1.18 dB	0.62 dB	0.42 dB	0.33 dB

Table 1: E_b/N_0 Loss vs Capacity to achieve $P_w(e) = 10^{-4}$ for different modulations and block size

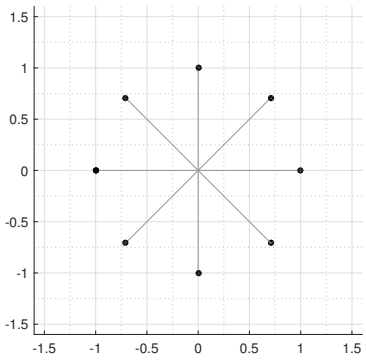


(a) 4-QAM constellation

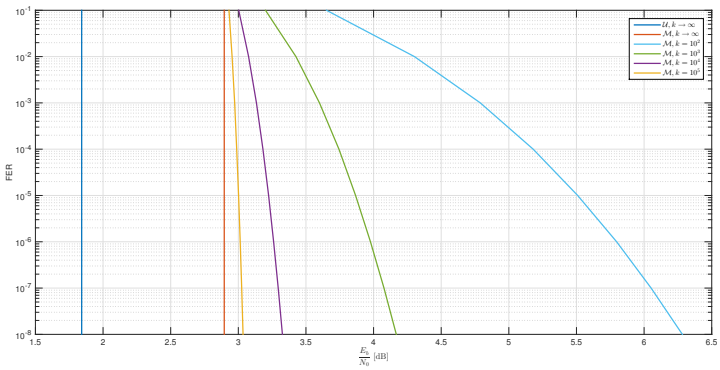


(b) Lower Bounds of FER

Figure 9: Lower Bounds of FER for 4-QAM with $r \simeq 2$, $k = 100, 1000, 1000, 10000, k \to \infty$.

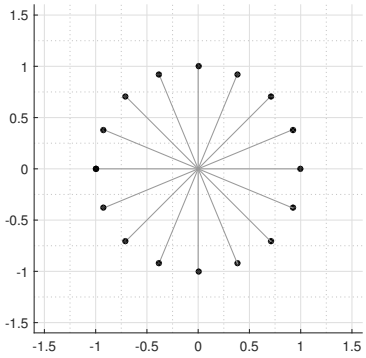


(a) 8-PSK constellation

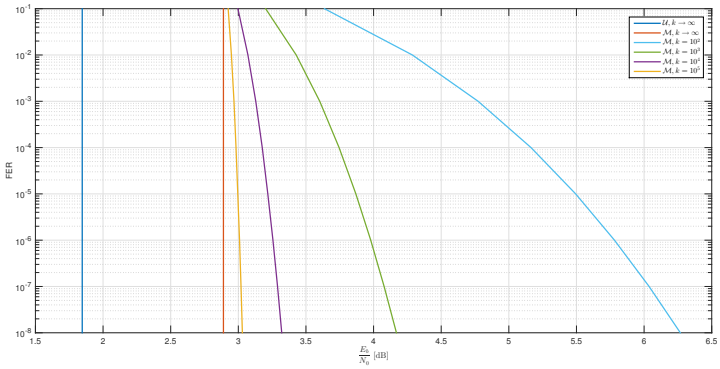


(b) Lower Bounds of FER

Figure 10: Lower Bounds of FER for 8-PSK with $r \simeq 2$, $k = 100, 1000, 1000, 10000, k \to \infty$.

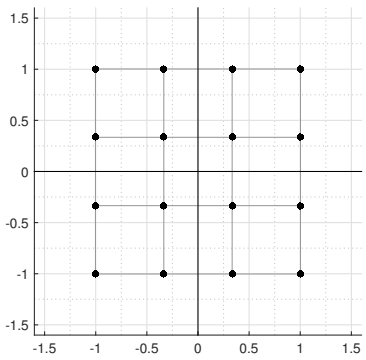


(a) 16-PSK constellation

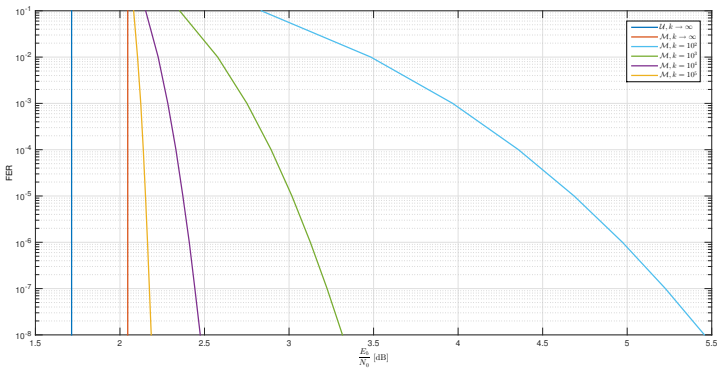


(b) Lower Bounds of FER

Figure 11: Lower Bounds of FER for 16-PSK with $r \simeq 2$, $k = 100, 1000, 1000, 10000, k \to \infty$.



(a) 16-QAM constellation



(b) Lower Bounds of FER

Figure 12: Lower Bounds of FER for 16-QAM with $r \simeq 2$, $k = 100, 1000, 1000, 10000, k \to \infty$.

PC Session 3: SISO Decoder

Purpose

1. To design and implement a C++ class realizing a Sliding Window with grouped decisions Soft-Input Soft-Output decoder (SWG-SISO), with input and output binary LLR.
2. To simulate the performance of a transmission system of Figure 1 employing a binary convolutional code, 2-PAM modulation, AWGN channel and a the developed binary SWG-SISO decoder.
3. To compute the BER performance at the input and at the output of the SISO decoder versus.

Problem Statement

1. Commented developed class (files **`SISO_Decoder.cpp`** and **`SISO_Decoder.h`**)
2. Simulation results relative to the rate $\frac{1}{2}$, 4 and 8 states convolutional encoder reporting the 4 BER measures versus the Es/N0 in [dB]

Available (new) software

1. Main program for the simulation that should be used for the
2. Declaration of the class **`SISO_Decoder`** and its empty implementation (**`SISO_Decoder.cpp`**)

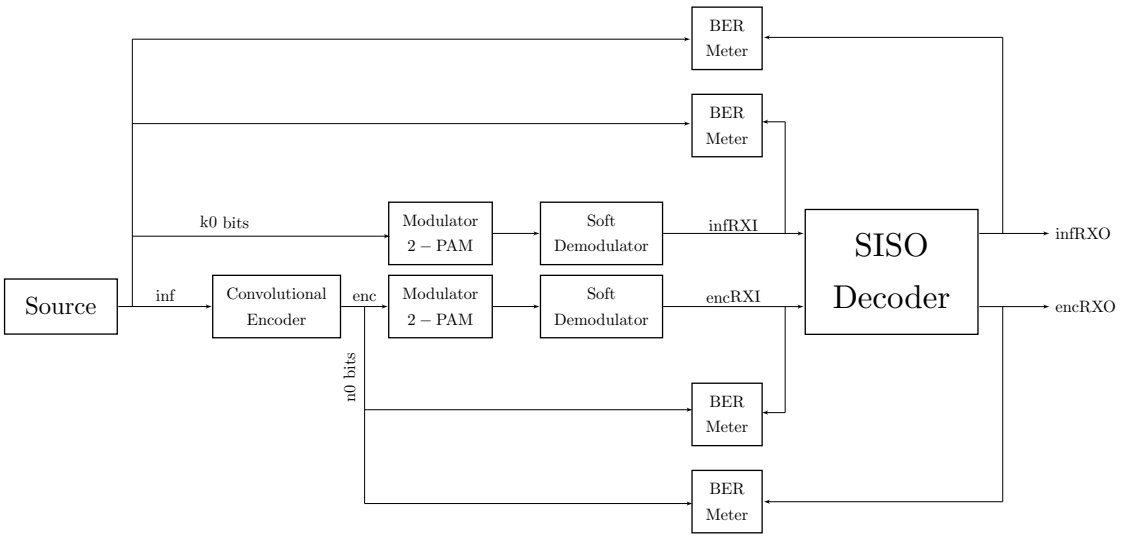


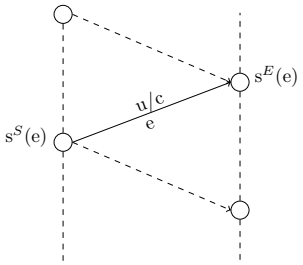
Figure 13: System Block Diagram

System Description

The aim of the PC session is to implement a SISO decoder by building a C++ class and to analyze the performances of the system computing the coding gain.
The block diagram of the simulated system is shown in Figure 21.
The simulater evolves in time frame by frame. For each frame the Source generates the information bits (`inf`). In order to compare the performances of the coded and uncoded transmitting system, we simulate two different AWGN channels with the same variance σ_N^2. The Convolutional Encoder takes k_0 information bits to generate the sequence `enc` of n_0 coded bits, so that its rate is $r_c = \frac{k_0}{n_0}$.
Both the sequences `inf` and `enc` are modulated with a 2-PAM modulation, then they are processed by the Soft Demodulator block, that emulates the channel behaviour adding noise samples, computes the log-likelihood ratio (LLR) sequences `infRXI` and `encRXI`.
The Soft Input Soft Output (SISO) Decoder elaborates the LLR input sequences `infRXI`, `encRXI` and provides the computed LLR output sequences `infRXO`, `encRXO`.
At the end, we perform hard demodulation on the LLR sequences, according to the Voronoi region rule for the 2-PAM constellation and estimate the BER before and after the SISO block, by counting the number of bits in error and dividing by the total number of transmitted bits.

The generic SISO module is a four-port device that, for the k-th Trellis Section takes as input the a priori LLRs of the uncoded and coded bits: $\lambda_k^u(I)$ and $\lambda_k^c(I)$ and provides the extrinsic a posteriori LLRs on uncoded and coded bits: $\lambda_k^u(O)$ and $\lambda_k^c(O)$.

The generic edge e of the Trellis section is uniquely identified by the starting state $s^S(e)$, the ending state $s^E(e)$ and the given label $\mathbf{u}/\mathbf{c}$ where $\mathbf{u}$ is the information bit sequence in input and $\mathbf{c}$ is the coded sequence in output from the Convolutional Encoder.



(a) Trellis Edge

(b) SISO Module

The basic idea of SISO decoding is to reconstruct the Trellis evolution in time of the Encoder.
We define the output branch metric $b_k(\mathbf{u}/\mathbf{c}, O)$ as the likelihood of being in a certain edge of the Trellis section at time instant k.

$$b_k((\mathbf{u}/\mathbf{c}), O) = \max_{e:\mathbf{u}(e)=\mathbf{u},\mathbf{c}(e)=\mathbf{c}}{}^{*} \alpha_{k-1}(s^S(e)) + b_k(\mathbf{u}(e)/\mathbf{c}(e), I) + \beta_k(s^E(s)) \tag{8}$$

where:

- $\alpha_{k-1}(s^S(e))$ is the likelihood of being in state $s^S(e)$ at time $k-1$ given the past observations (Forward Path Metric);
- $\beta_k(s^E(e))$ is the likelihood of being in state $s^E(e)$ at time k given the future observations (Backward Path Metric);
- $b_k(\mathbf{u}(e)/\mathbf{c}(e), I)$ is the input branch metric and depends on the input LLRs.

The Forward and Backward Path Metrics expressing the likelihood of being in the state s at time instant k are computed recursively by:

$$\alpha_k(s) = \max_{e:s^E(e)=s}{}^{*} \left[\alpha_{k-1}(s^S(e)) + b_k(\mathbf{u}(e)/\mathbf{c}(e), I)\right]$$

$$\beta_k(s) = \max_{e:s^S(e)=s}{}^{*} \left[\beta_{k+1}(s^E(e)) + b_{k+1}(\mathbf{u}(e)/\mathbf{c}(e), I)\right] \tag{9}$$

where $b_k(\mathbf{u}(e)/\mathbf{c}(e), I)$ is the input branch metric computed by:

$$b_k(\mathbf{u}(e)/\mathbf{c}(e), I) = \lambda_k(\mathbf{u}(e), I) + \lambda_k(\mathbf{c}(e), I) \tag{10}$$

$\lambda_k(\mathbf{u}(e), I)$ and $\lambda_k(\mathbf{c}(e), I)$ are the likelihood of the symbols $\mathbf{u}/\mathbf{c}$ computed by aggregation combining the likelihood of the constituent bits:

$$\lambda_k(\mathbf{u}, I) = \sum_{i=0}^{k_0-1} u_i \lambda^u_{(kk_0+i)} \qquad\qquad \lambda_k(\mathbf{c}, I) = \sum_{i=0}^{n_0-1} c_i \lambda^c_{(kn_0+i)} \tag{11}$$

From the output branch metrics computed by (8), we obtain the LLRs on symbols. In order to obtain LLRs on bits we need to use marginalization:

$$\lambda^u_{kk_0+i}(O) = \max_{\mathbf{u}:u_i=1}{}^{*} b_k(\mathbf{u}/\mathbf{c}, O) - \max_{\mathbf{u}:u_i=0}{}^{*} b_k(\mathbf{u}/\mathbf{c}, O) - \lambda^u_{kk_0+i}(I)$$

$$\lambda^c_{kn_0+i}(O) = \max_{\mathbf{c}:c_i=1}{}^{*} b_k(\mathbf{u}/\mathbf{c}, O) - \max_{\mathbf{c}:c_i=0}{}^{*} b_k(\mathbf{u}/\mathbf{c}, O) - \lambda^c_{kn_0+i}(I) \tag{12}$$

In line of principle, the SISO may be run to decode the entire sequence in one-shot, but this induces a huge requirement of memory and delay.

The solution to this is the adoption of the Sliding Window with Grouped Decisions SISO (SWG-SISO). The SWG-SISO decodes the sequence by splitting it into blocks of length N_{bl}.

The backward recursion at time k requires the knowledge of all the backward path metrics at time $> k$. In order to solve this, we initialize the backward path metrics at time $N_{bl}+D-1$ with a uniform distribution $1/N$ where $N = 2^\nu$ is the number of states.

If D is sufficiently large ($D > 6 - 7\nu$) independently from starting condition of backward recursion, the algorithm converges to the true result.

In order to decode N_{bl} Trellis sections, we first perform backward recursion on $N_{bl} + D - 1$ steps, then we proceed with N_{bl} forward recursion steps, as shown in Figure 15.

At the end of the decoding of the actual block, we shift the window of N_{bl} in order to decode the next block.

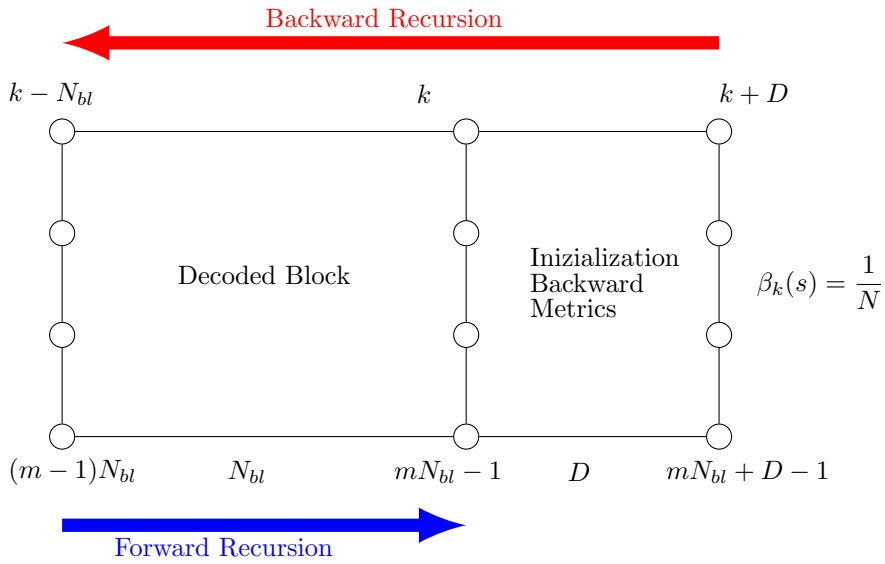


Figure 15: SWG-SISO

SISO Implementation

The implementation of the SWG-SISO Decoder is made by building a C++ Class. The Aggregation and Marginalization are implemented respectively by the methods **`BitToBranch`** and **`BranchToBit`**.
The code of the `BitToBranch` method is the following:

```
void SISO_Decoder::BitToBranch(const int* inp)
{
  int temp,k,l,s;
  s=1;
  for(k=0;k<nbit;k++)
  {
    temp = branch[s] = *inp++;
    for(l=1;l<s;l++)branch[s+l]=temp+branch[l];
    s *=2;
  }
}
```

The array `branch` contains the input LLRs on symbols obtained by aggregation of bits LLRs of the input `inp`.

The aggregation is performed in an efficient way.
The value `nbit=k0+n0` is the number of bits that form a symbol.

The following pseudocode shows the aggregation for `k0=1`, `n0=2`:

`branch[0]=`λ`(000)=0;`

`s=1;`

`branch[1]=`λ`(100)=`λ^u`;`

`s=2;`

`branch[2]=`λ`(010)=`λ^{c_1}`;`

`branch[3]=branch[2]+branch[1]=`λ`(110)=`$\lambda^{c_1} + \lambda^u$`;`

`s=4;`

`branch[4]=`λ`(001)=`λ^{c_2}`;`

`branch[5]=branch[4]+branch[1]=`λ`(101)=`$\lambda^{c_2} + \lambda^u$`;`

`branch[6]=branch[4]+branch[2]=`λ`(011)=`$\lambda^{c_2} + \lambda^{c_1}$`;`

`branch[7]=branch[4]+branch[3]=`λ`(111)=`$\lambda^{c_2} + \lambda^{c_1} + \lambda^u$`;`

The code of the `BranchToBit` method is the following:

```
void SISO_Decoder::BranchToBit(int* out)
{
  int k,j,n0,n1;
  int Nu=nlab;
  for(j=nbit-1;j>0;j--)
  {
    Nu>>=1;
    n0=brancho[0];
    for(k=1;k<Nu;k++)
      maxx(n0,brancho[k]);
    n1 = brancho[Nu];
    maxx(brancho[0],brancho[Nu]);
    for(k=1;k<Nu;k++)
    {
      maxx(n1,brancho[Nu+k]);
      maxx(brancho[k],brancho[Nu+k]);
    }
    out[j]=n1-n0;
  }
  out[0]=brancho[1]-brancho[0];
}
```

The array `brancho` contains the output LLRs on symbols that form the labels of each branch, `nlab` is the number of possible labels equal to 2^{nbit}, `out` is a vector containing the result of the marginalization for each bit.

The following pseudocode shows the aggregation for `k0=1`, `n0=2`:

`Nu=nlab=`2^{nbit}`;`

`j=2;`

`Nu=4;`

`n0=max*(brancho[0],brancho[1],brancho[2],brancho[3])`

`=max*(`λ`(000),`λ`(100),`λ`(010),`λ`(110));`

`n1=max*(brancho[4],brancho[5],brancho[6],brancho[7])`

`=max*(`λ`(001),`λ`(101),`λ`(011),`λ`(111));`

`brancho[0]=max*(brancho[0],brancho[4])=max*(`λ`(000),`λ`(001))=`λ`(00-);`

`brancho[1]=max*(brancho[1],brancho[5])=max*(`λ`(100),`λ`(101))=`λ`(10-);`

`brancho[2]=max*(brancho[2],brancho[6])=max*(`λ`(010),`λ`(011))=`λ`(01-);`

`brancho[3]=max*(brancho[3],brancho[7])=max*(`λ`(110),`λ`(111))=`λ`(11-);`

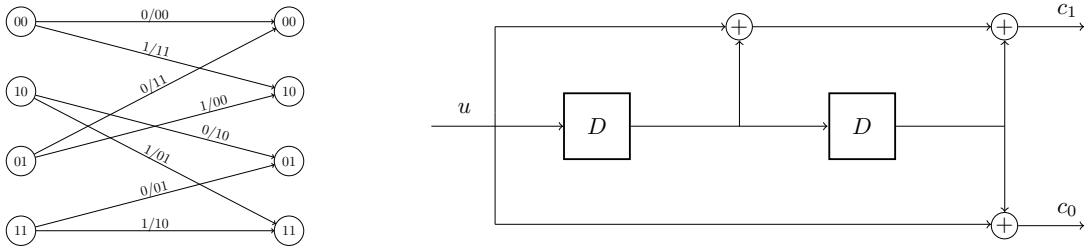
`out[2]=n1-n0;`

```
j=1;
Nu=2;
n0=max*(brancho[0],brancho[1])=max*(λ(00-),λ(10-));
n1=max*(brancho[2],brancho[3])=max*(λ(01-),λ(11-));
brancho[0]=max*(brancho[0],brancho[2])=max*(λ(00-),λ(01-))=λ(0--);
brancho[1]=max*(brancho[1],brancho[3])=max*(λ(10-),λ(11-))=λ(1--);
out[1]=n1-n0;
out[0]=brancho[1]-brancho[0];
```

This implementatation of marginalization is recursive. The branch metrics are divided into groups: one with the last bit equal to 1, the other with the last bit equal to 0 and we compute the output by using the $\max^*$ operator. The procedure is iterated on the second and first bit. By doing this, we implement a recursion that allows to reduce the computational complexity from $\mathcal{O}(N \cdot 2^N)$, applying the Equation (12) to $\mathcal{O}(3 \cdot 2^N)$ where N is the number of labels.
The output LLRs on bits are obtained by subtracting from `out` the corresponding input LLR on bit.

Results Discussion

The simulation is performed using 4 and 8 states Non Systematic Non Recursive (NSNR) Convolutional Encoder with rate $r = \frac{1}{2}$.
The encoder representations and the relative Trellis Diagrams are shown in Figure 16 and 17.
The chosen Parameter for the SWG-SISO are $D = 20$ and $N_{bl} = 1000$.

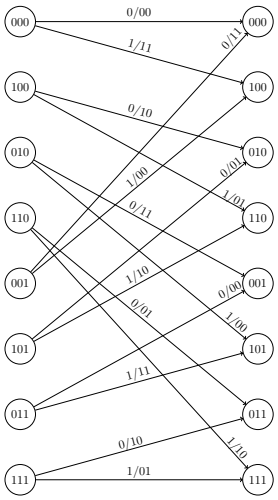


(a) Trellis Diagram

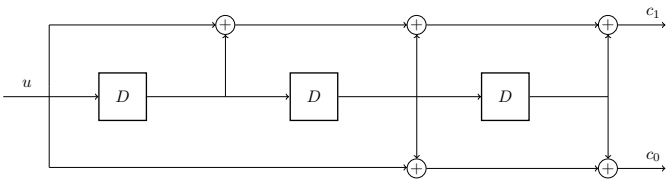
(b) Convolutional Encoder Structure

Figure 16: 4 state NSNR Encoder, $r = \frac{1}{2}$, $G = [1 + D + D^2, 1 + D^2]$

In Figure 18 we plot the BER curves of the information and encoded bits before and after the SISO block obtained taking hard decision based on the sign of the LLRs sequences `infRXI`, `encRXI`, `infRXO` and `encRXO`. As we expect the BER performances before SISO are the same for both the information and encoded bits. The slight difference at high E_b/N_0 values is due to the insufficient statistics of counted errors. After the SISO the information bits error rate is lower than encoded ones since with $r = \frac{1}{2}$ only half of errors affect the information bits.



(a) Trellis Diagram



(b) Convolutional Encoder Structure

Figure 17: 8 state NSNR Encoder, $r = \frac{1}{2}$, $G = [1 + D + D^2 + D^3, 1 + D^2 + D^3]$

In Figure 19 we plot the Information Bit Error curves as function of E_b/N_0 for 4 and 8 state Encoders before and after the SISO. As we expect, increasing the constraint length of the encoder the performances improve since the correction capability of the Convolutional Encoder is strictly related to its memory.
In Figure 20 we plot the coding gain for different values of BER for both the two encoders computed as the difference between E_b/N_0 required to achieve a target BER value before and after the SISO. As we expect, the coding gain becomes almost constant decreasing the BER target. The 4 and 8 State Encoders reach respectively 6.8 dB and 8 dB of gain.

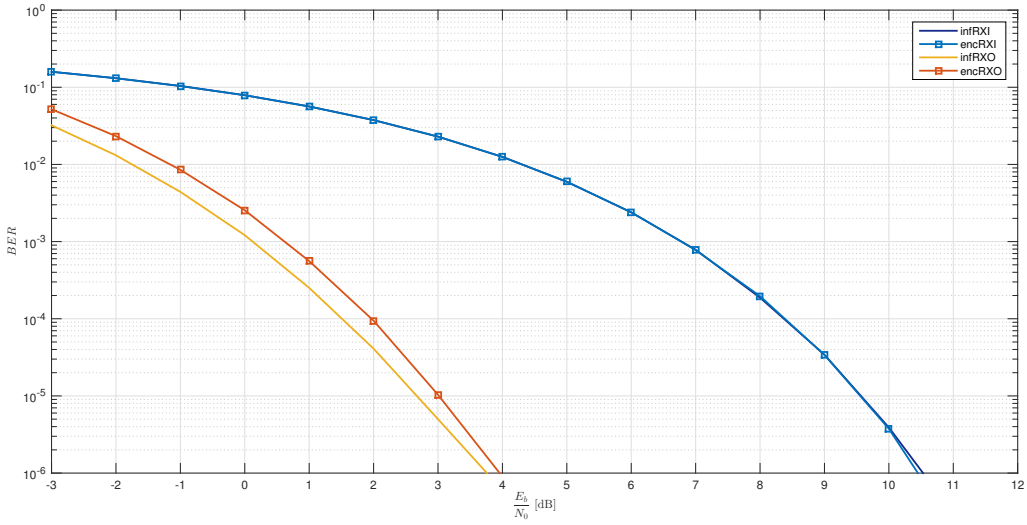


Figure 18: BER curves before and after SISO for 4 state Encoder.

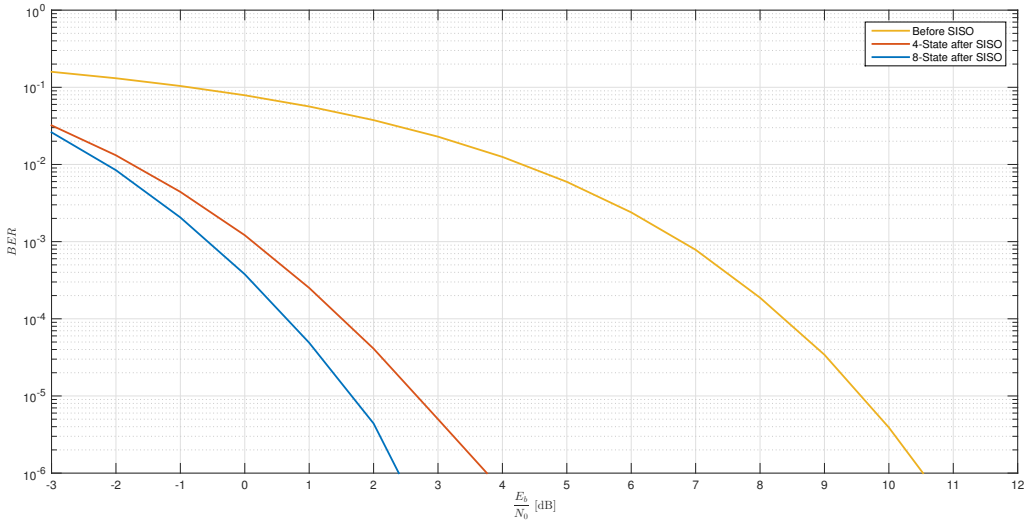


Figure 19: Information BER before and after SISO for 4 and 8 state Encoders.

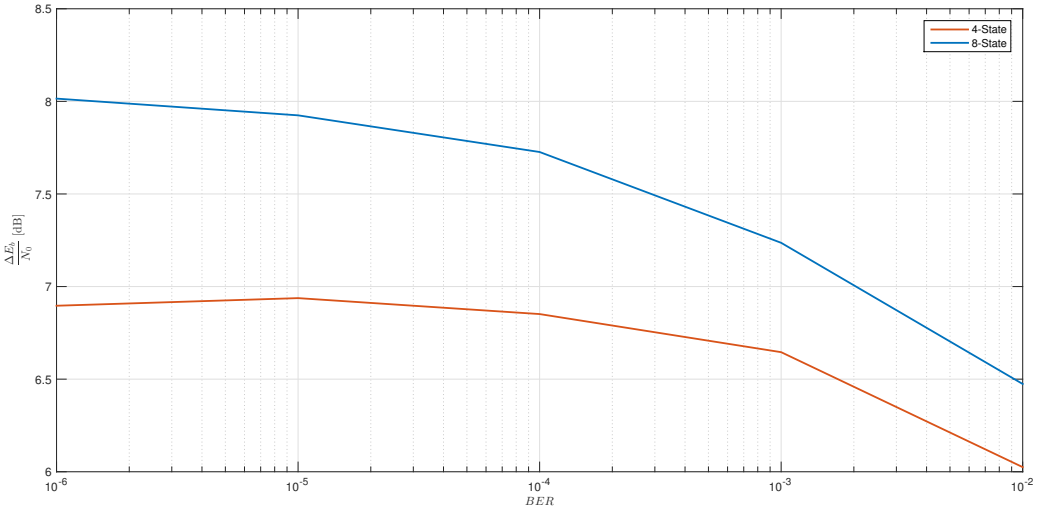


Figure 20: Coding gains of 4 and 8 States Convolutional Codes varying the BER target.

PC Session 4: PCCC Decoder

Purpose

1. To understand the simulation program for a PCCC.
2. To understand the termination of convolutional codes.
3. To simulate and plot the performance of a rate 1/3 PCCC with different constituent encoders and interleaver block sizes.

Problem Statement

1. BER and FER plot with 4 and 8 state encoders and interleaver. size 100, 1000 and 10000
2. Comment on the plots

Available (new) software

1. Main program for the simulation that should be used for the simulation of PCCC
2. Interleaver class and a new version of **`SISO_Decoder`**.

System Description

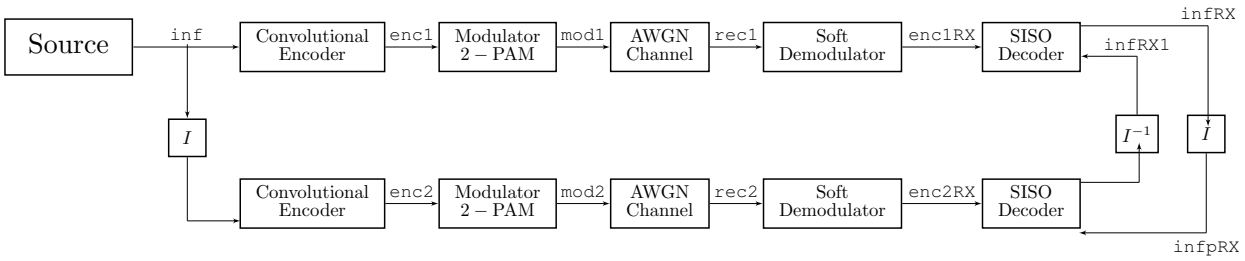


Figure 21: System Block Diagram

The aim of the laboratory session is to analyze the performances of communication system using a Parallel Concatenated Convolutional Code (PCCC). The system block diagram is shown in Figure 21.
The Source generates a sequence of `inf` information bits of length N.

The PCCC Encoder is composed of two constituent Convolutional Encoders working in parallel: The Upper Encoder generates the sequence `enc1` of encoded bits with rate r_1. The information sequence is passed to

an Interleaver, then the Lower Encoder generates the sequence `enc2` with rate r_2.
The overall PCCC Encoder has rate $r = \dfrac{r_1 \cdot r_2}{r_1 + r_2}$.

The two sequences `enc1` and `enc2` are both mapped by a 2-PAM modulation scheme, to produce the symbol sequences `mod1` and `mod2` that are transmitted over an AWGN Channel. The received sequences are `rec1`, `rec2` are passed to Soft Demodulators that produces LLRs sequences on coded bits: `enc1RX`, `enc2RX`.

The two sequences `enc1RX` and `enc2RX` are processed by the PCCC Decoder to extrapolate the information bits.

The PCCC Decoder is an Iterative Decoder composed by two SISO Decoders that share informations about the LLRs on output information bits.
The extrinsic information computed from one encoder is considered as a-priori information for the other one and vice-versa.

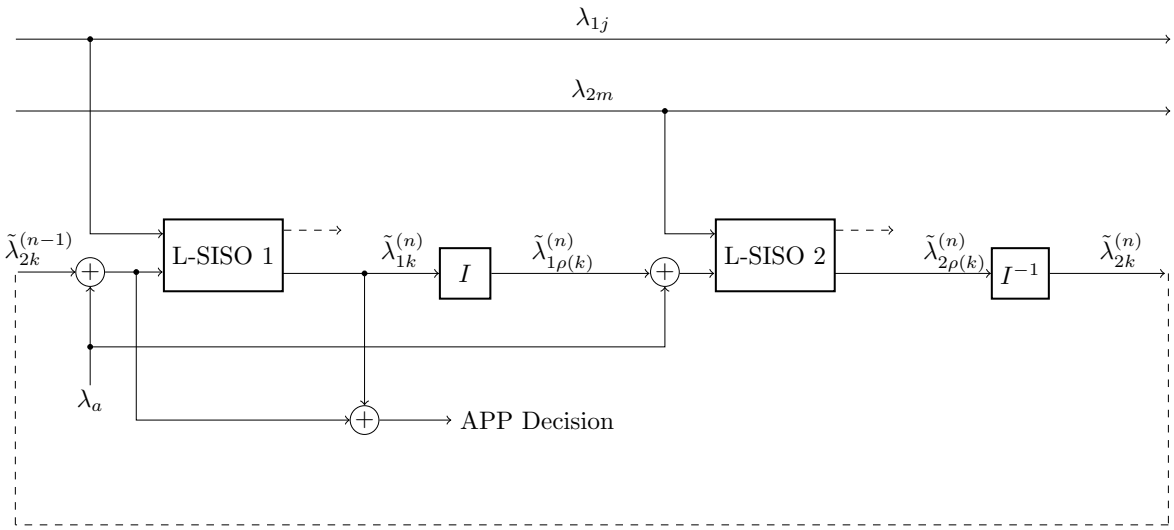


Figure 22: Iterative SISO Decdoder

The Decoding Procedure for the generic iteration of index n is the following:

- The Upper SISO Decoder takes as input:
  - $\lambda_{1j} =$ `enc1RX` LLRs on coded bits coming from the Soft Demodulator;
  - λ_a A Priori LLRs;
  - $\tilde{\lambda}_{2k}^{(n-1)} =$ `infRX1` Extrinsic LLRs from the Lower SISO Decoder at the previous iteration.
- The input sequences are used to update the extrinsic LLRs sequence on information bits: $\tilde{\lambda}_{1k}^{(n)} =$`infRX`.
- The output sequence $\tilde{\lambda}_{1k}^{(n)}$ is interleaved producing the sequence $\tilde{\lambda}_{1\rho(k)}^{(n)} =$`infpRX`.
- The Lower SISO Decoder takes as input:

- λ_{2m} = `enc2RX` LLRs on coded bits coming from the Soft Demodulator;
- λ_a A Priori LLRs;
- $\tilde{\lambda}^{(n)}_{1\rho(k)}$ = `infpRX` Extrinsic LLRs from the Upper SISO Decoder.

- The input sequences are used to update the extrinsic LLRs sequence on information bits: $\tilde{\lambda}^{(n)}_{2\rho(k)}$.
- The resulting sequence $\tilde{\lambda}^{(n)}_{2\rho(k)}$ is de-interleaved producing the output $\tilde{\lambda}^{(n)}_{2k}$ =`infRX1`.

The above operations are repeated iterating between the two SISO Decoders.

At the first iteration, the upper SISO Decoder has no extrinsic informations from the Lower SISO Decoder and so the sequence is initialized to zero: $\tilde{\lambda}^{(0)}_{2k} = 0$, under uniform distribution assumption.

At the end of iterations, a Hard Decision is performed according to the sign of the A Posteriori (APP) extrinsic LLRs sequence.

$$\lambda_k^{(n)^{(\mathrm{APP})}} = \tilde{\lambda}^{(n)}_{1k} + \tilde{\lambda}^{(n)}_{2k} + \lambda_a \tag{13}$$

PCCC Decoder Implementation

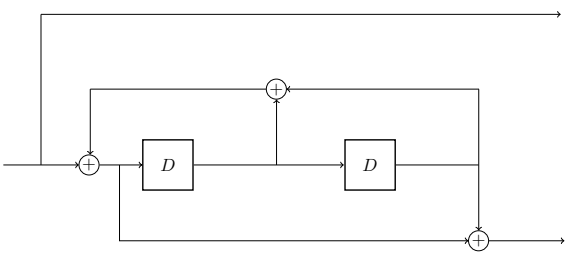
The code that implements the iterative PCCC Decoder is shown in Listing 1.

```
for(nit=0;nit<niter;nit++)
{
  if(nit==0)
    SISODec1  ->RunBlock(0,enc1RX,infRX,0);
  else
    SISODec1  ->RunBlock(infRX1,enc1RX,infRX,0);

  Interl->Run(1,infRX,infpRX);

  SISODec2  ->RunBlock(infpRX,enc2RX,infpRX,0);
  Interl->Run(1,infpRX,infRX1,false);

  for(i=0;i<k0*N;i++)infRX[i]+=infRX1[i];
```

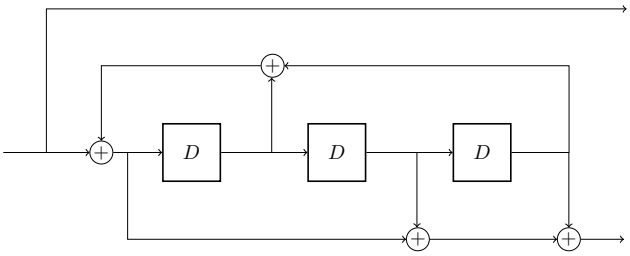
Listing 1: Code Implementing PCCC Decoder

Results Discussion

We analyze the performances of PCCC Decoder with rate $r = \frac{1}{3}$, Systematic Recursive constituent encoders of $4, 8$ states, interleavers of size $100, 1000, 10000$.
The generic Systematic Recursive (SR) Encoder is characterized by Feed-Forward and Backward polynomials whose coefficients are represented in octal basis by the pair $[Z, H]$.
The constituent convolutional encoders used in the simulations are shown in Figure 23a, 23b.
For the 4 states SR encoder we use $[Z, H] = [05, 07]$, for the 8 states SR encoder we use $[Z, H] = [015, 013]$.
We compare the loss induced by the use of max operator instead of max^* operator.
The $max^*(\cdot)$ operator is implemented by adding a correction factor obtained by a Look-Up Table (LUT) to the value assumed by $max(\cdot)$ as shown in Figure 24. The dimension and the values of LUT depends on the precision p. We set $p = 3$ and for this value of p we implement.
In Figure 25 we plot the BER performances by varying the interleaver size and using, respectively, $max(\cdot)$ and $max^*(\cdot)$ operator. We can see that the BER curves improve when we increase the interleaver size. The



(a) 4 states, $r = \frac{1}{2}$, $G = \left[1, \frac{1+D^2}{1+D+D^2}\right]$



(b) 8 states, $r = \frac{1}{2}$, $G = \left[1, \frac{1+D^2+D^3}{1+D+D^3}\right]$

Figure 23: Constituent SR Encoders

drawback is the latency of the decoder.

We can recognize the two main regions of a code: the Waterfall and the Error Floor region. In the first region the BER curve decreases very rapidly with E_b/N_0, while in the second we do not obtain great improvement increasing E_b/N_0.

Comparing performances of 4 and 8 states encoders we can not say that one performs better than the other for all values of E_b/N_0. There is a point where the two curves cross each other. This implies that one encoder is better than the other for lower values of E_b/N_0, and vice versa for higher values.

In particular, the Waterfall Region of 4-State encoder is moved to the left with respect to the 8-State's one. However, its Error Floor Region occurs for a higher value of BER. This implies the crossing of the two curves. In line of principle we aim to design a code that presents a Waterfall Region moved to the left as much as possible, and an Error Floor region as low as possible, but PCCC is characterized by a higher Error Floor Region when Waterfall Region moves to the left.

For large values of E_b/N_0, performances are better increasing the number of states of constituent encoders. However, the gain obtained increasing the number of states is smaller with respect to that one obtained increasing the interleaver size (BER $\propto 1/N$).

In Figure 26 we plot the FER performances by varying the interleaver size using, respectively, $max(\cdot)$ and $max^*(\cdot)$ operator. We can recognize the same characteristics of BER curves.

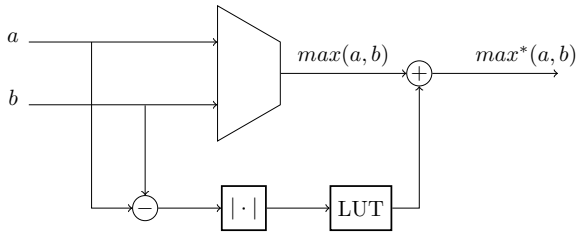
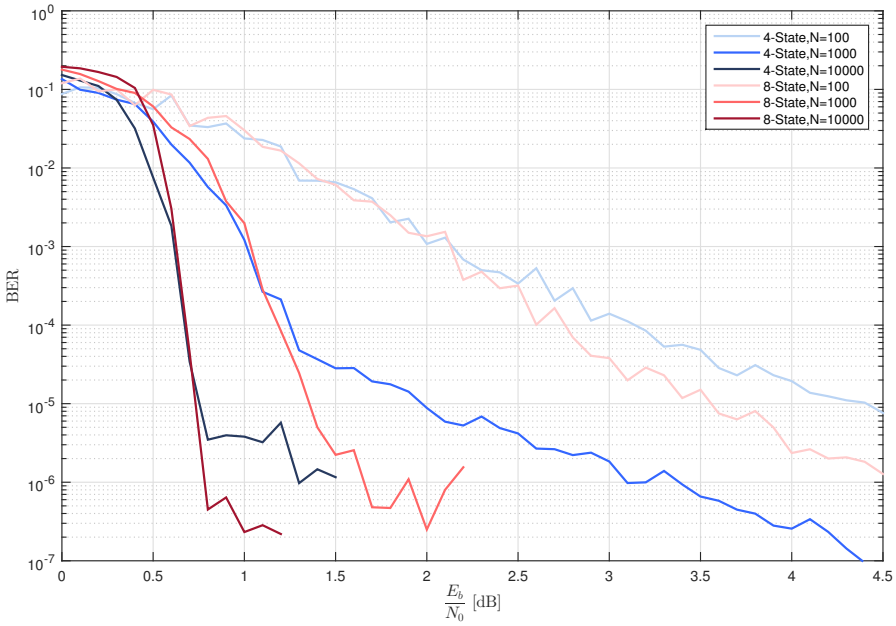


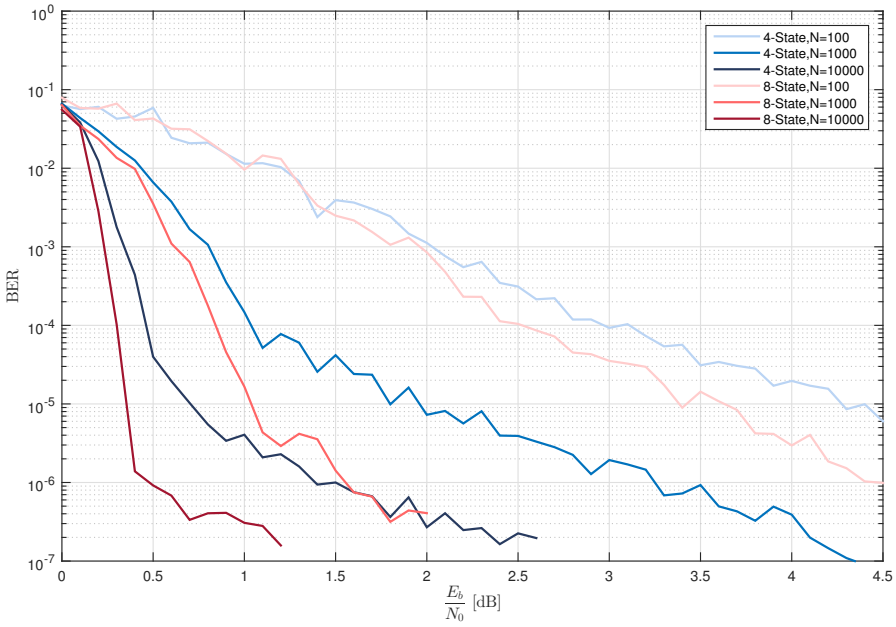
Figure 24: $max^*(\cdot)$ operator logical scheme.

In Figure 27 we compare the BER curves using the two different operators. The Interleaver size is set to $N = 10000$. The approximation of $max^*(\cdot)$ with $\max(\cdot)$ is suboptimal since performances are worse. The loss introduced by the $max(\cdot)$ is less or equal to 0.5 dB and it increases if the number of states of Constituent Encoders increases.

For large values of E_b/N_0 we can notice that the performances are almost identical. When E_b/N_0 is high the probability of having paths with close metric in the Forward and Backward recursions is low, then the

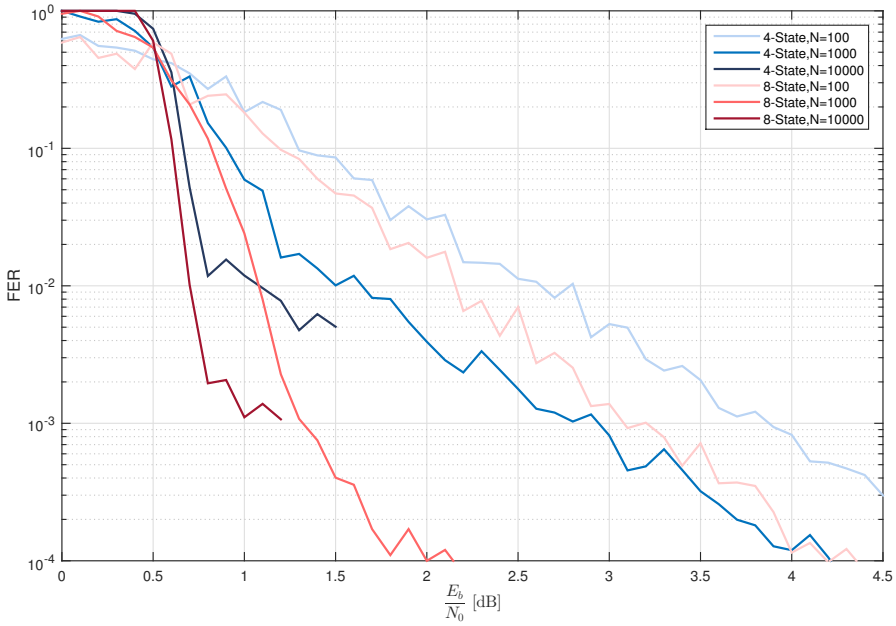


(a) $max(\cdot)$

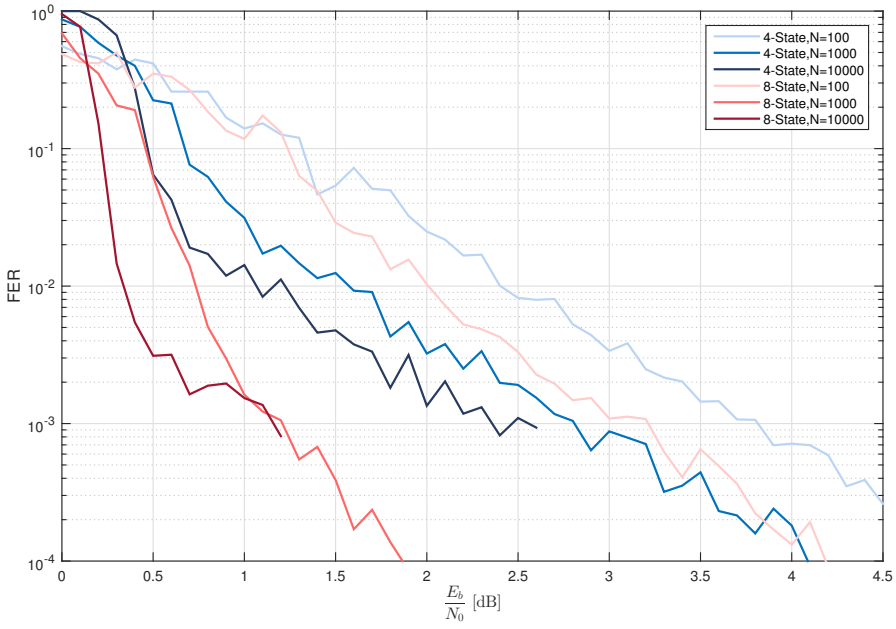


(b) $max^*(\cdot)$

Figure 25: BER curves at varying of interleaver size



(a) $max(\cdot)$



(b) $max^*(\cdot)$

Figure 26: FER curves at varying of interleaver size

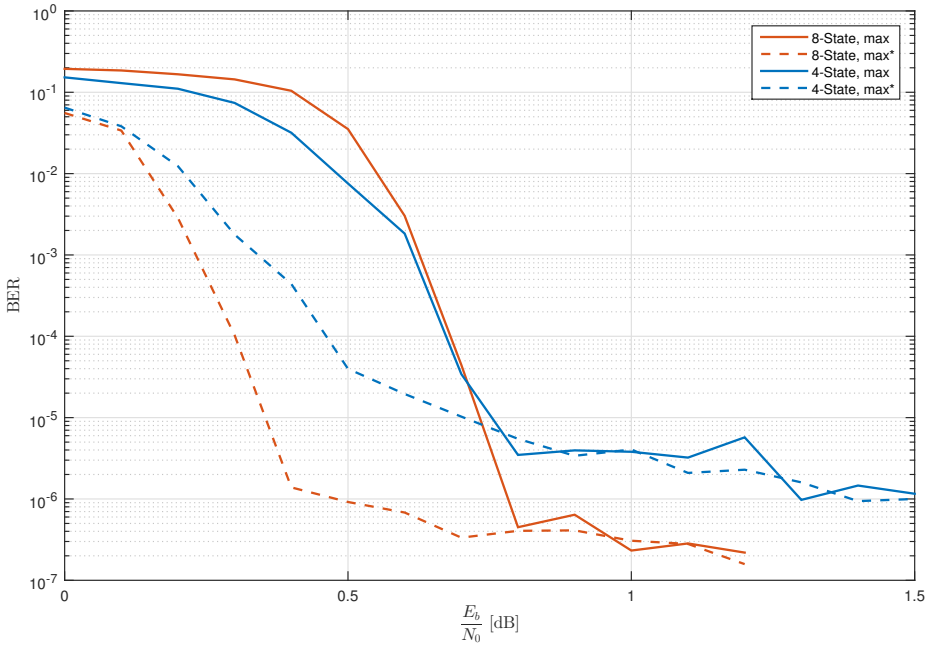


Figure 27: Comparison of BER performances using $max(\cdot)$ and $max^*(\cdot), N = 10000$

correction factor is almost zero and the two operators yield the same performances. Therefore the use of $max^*(\cdot)$ improves performances only for low E_b/N_0.

All the results shown until now, are obtained using a number of iterations in PCCC decoder equal to 10. Then, we investigate the relation between performances and number of decoder iterations.

In Figure 28 we plot BER versus the number of iterations using 4-State Constituent Encoders with interleaver size $N = 1000$ fixing the value of E_b/N_0 for each curve.

We can see that for low number of iterations, we reach big improvements increasing it. For high number of iterations, by further increasing the number of iterations, performances do not improve since the Decoder does not change decision from one iteration to the other. In this case, it is time consuming to keep the Decoder getting stuck in the loop. We can adopt as solution to exit from the loop when the Decoder's decision does not change for every bit of the information sequence, instead of using a fixed number of iterations.

Finally, we point out that PCCC performances depend on the characteristics of Constituent Encoders. PCCC adopts SR Encoders even if they are worse with respect to NSNR Encoders in terms of free distance d_{free}. However, the Interleaver gain depends on the effective free distance $d_{eff,free}$ and so SR yield better performances. The SR Constituent Encoders must be designed in order to maximize this parameter. If the Convolutional Encoder has a primitive polynomial associated to the Backward part, the distance is increased and the performances are better.

In Figure 29 we compare the FER performances of PCCC with $N = 10000$ using the 4-State Constituent Encoder of Figure 23a and the 4-State inverting the Forward and Backward polynomials. The first has a Primitive Backward Polynomial $D^2 + D + 1$, while the second has not since $D^2 + 1 = (D + 1)^2$ over the

Galois Fields of order 2. We can notice that the use of not Primitive Polynomials gets worse performances.

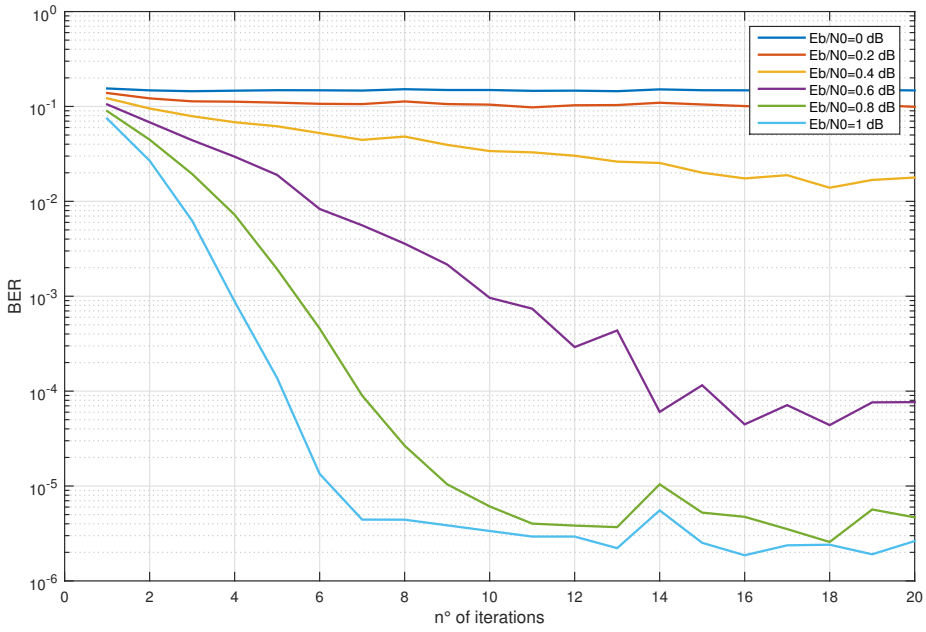


Figure 28: Performances at varying of number of decoder iterations

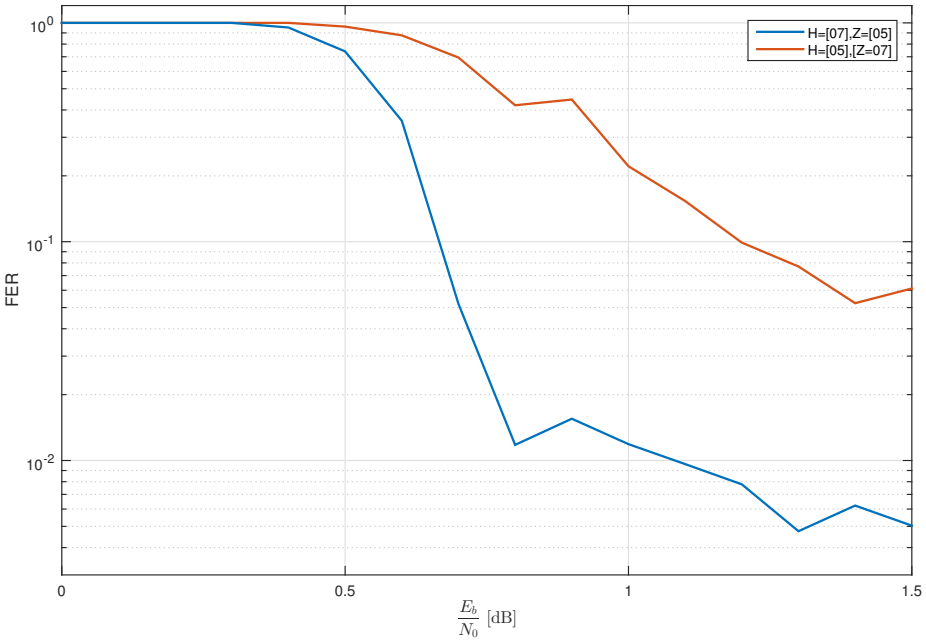


Figure 29: FER performances comparing the use of Primitive Polynomials

PC Session 5: LDPC Decoder

Purpose

1. To understand the simulation program for an LDPC
2. To understand LDPC encoding by back substitution
3. To understand the difference between optimal and suboptimal decoding algorithm (min-sum-offset) for LDPC
4. To simulate and plot the performance of the LDPC code for the DVB-S2 standard.

Problem Statement

1. BER and FER plot with rate 1/4 and 8/9 with medium block length (16200) with optimal and sub-optimal decoder.
2. BER plot with rate 1/2 with optimal decoder and suboptimal decoder for different values of the offset parameter beta (short block).
3. BER plot with rate 1/2 and the three values of available block length (short, medium and long code) use the algorithm that you like.
4. Comment on the plots

Available (new) software

1. Main program for the simulation that should be used for the simulation of LDPC
2. **`LDPC_Encoder`** and **`LDPC_Decoder`** class

System Description

The aim of this laboratory is to evaluate the performances of Low Density Parity Check Codes (LDPC) using DVB-S2 Standard.
LDPC Codes are a particular class of Linear Block Codes whose parity-check matrix $\mathbf{H}$ is sparse.
The block diagram of the simulation chain is reported in Figure 30.
The LDPC Encoder takes as input the information sequence `inf`, and for each block of k information bits it generates blocks of n bits composing the sequence `enc`.
Since $\mathbf{H}$ is sparse, an efficient representation is to indicate the indices of ones for each row.
Here is reported an example for the suggested representation of $\mathbf{H}$:

$$\mathbf{H} = \left[\begin{array}{cc|ccc} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} & h_{1,5} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & h_{2,5} \\ h_{3,1} & h_{3,2} & h_{3,3} & h_{3,4} & h_{3,5} \end{array}\right] = \left[\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array}\right] \rightarrow \begin{array}{l} h_1 = [\ 1,\ 2,\ 3\ ] \\ h_2 = [\ 2,\ 3,\ 4\ ] \\ h_3 = [\ 1,\ 3,\ 5\ ] \end{array} \tag{14}$$

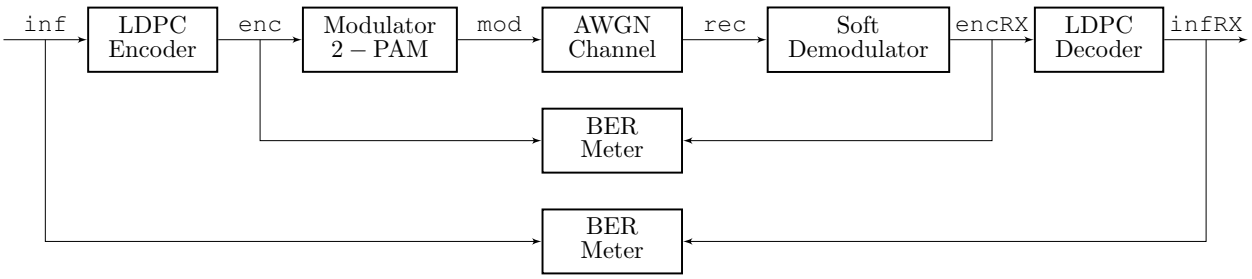


Figure 30: Block Diagram of LDPC Encoding scheme.

Since the Encoder is systematic, it propagates the k information bits and appends the computed $(n-k)$ parity check bits by applying the back substitution technique.
The technique provides a simple linear encoding procedure that can be applied to lower triangular parity-check matrices.
Each parity check bit is computed sequentially as linear combination of information bits and parity check bits computed at the previous steps.
The back substitution applied to the matrix $\mathbf{H}$ is the following:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 + x_2 \\ x_2 + c_3 \\ x_1 + c_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 + x_2 \\ x_2 + x_1 + x_2 \\ x_1 + x_1 + x_2 \end{bmatrix} \tag{15}$$

As we can see, we have to compute the value of c_3 in order to compute the values of c_4 and c_5.
The advantage of using the back substitution is reducing the complexity from $O(N^2)$ using the standard matrix product to $O(N)$.

LDPC Implementation

The code that implements the encoding technique is the following:

```
h=Hconn;
      for(i=0;i<K;i++)out[i]=inp[i];
      for(;i<N;i++)
      {
        temp=0;
        for(j=1;j<ncheck;j++)
        {
          if(h[j]==-1)break;
          temp ^= out[h[j-1]];
        }
        out[h[j-1]]=temp;
        h+=ncheck;
      }
      break;
```

The output codewords are sent over an AWGN Channel and the Soft Demodulator provides LLRs on the coded bits, wich are used by the LDPC Decoder to decode the information bit sequences.
The Decoding rule is based on belief propagation algorithm between the variable and check nodes that exchange messages. The algorithm can be described graphically by constructing the Tanner Graph associated

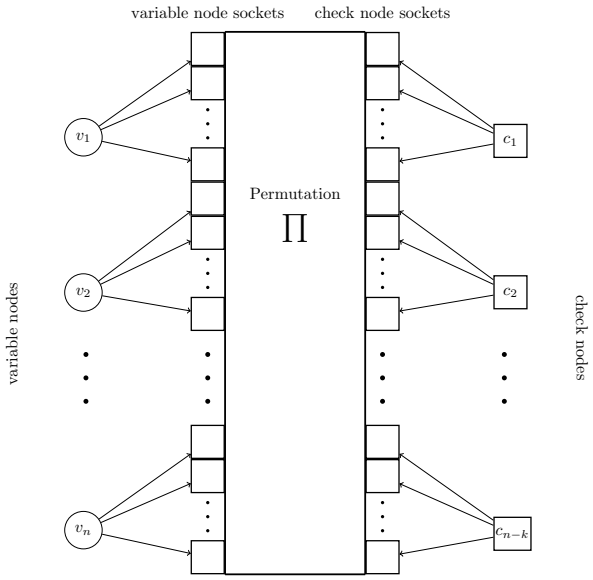


Figure 31: Tanner Graph.

to the **H**. In Figure 31 is reported a Tanner Graph for a generic LDPC, where $\{v_1, ..., v_n\}$ are the variable nodes and $\{c_1, \cdots, c_{n-k}\}$ are the check nodes.

The number of edges connected to a variable node v_i is the degree of the variable node d_{v_i} , whereas the number of edges connected to a check node c_i is the degree of the check node d_{c_i}. At the n^{th} iteration of the algorithm, the generic message sent by a variable node v_i to a check node c_j is denoted by $\alpha_{ij}^{(n)}$, whereas the message sent by a check node c_j to a variable node v_i is denoted by $\beta_{ji}^{(n)}$ and they are stored into relative buffers called sockets.

Adopting the belief propagation algorithm, $\alpha_{ij}^{(n+1)}$ is computed as:

$$\alpha_{ij}^{(n+1)} = \lambda_i + \sum_{l=1, l\neq j}^{d_{v_i}} \beta_{li}^{(n)} \tag{16}$$

While the computation of $\beta_{ji}^{(n+1)}$ can be done in two different ways: the Sum-Prod and the Min-Sum. The first is optimal in terms of correction capability while the second is sub-optimal but it is less complex in terms of computation.

Sum-Prod - The Sum-Prod updating rule for $\beta_{ji}^{(n)}$ is the following:

$$\beta_{ji}^{(n+1)} = 2 \cdot \operatorname{arctanh}\left(\prod_{l=1, l\neq i}^{d_{c_j}} \tanh\left(\frac{\alpha_{lj}^{(n)}}{2} \right) \right) \tag{17}$$

The previous equation is equivalent to the $g(\cdot)$ *operator* defined as:

$$g(a, b) = \max^{*}(a, b) - \max^{*}(a + b, 0) \tag{18}$$

whose block diagram is reported in Figure 32.

Min-Sum - The Min-Sum updating rule for $\beta_{ji}^{(n)}$ is the following:

$$\beta_{ji}^{(n+1)} = - \prod_{l=1, l\neq i}^{d_{c_j}} \operatorname{sign}\left(-\alpha_{lj}^{(n)} \right) \cdot \min_{l=1, l\neq i}^{d_{c_j}} \left(|\alpha_{lj}^{(n)}| \right) \tag{19}$$

which is based on the following approximation of $g(\cdot)$ *operator*:

$$g(a,b) \simeq \text{sign}(a \cdot b) \cdot \min(|a|, |b|) \tag{20}$$

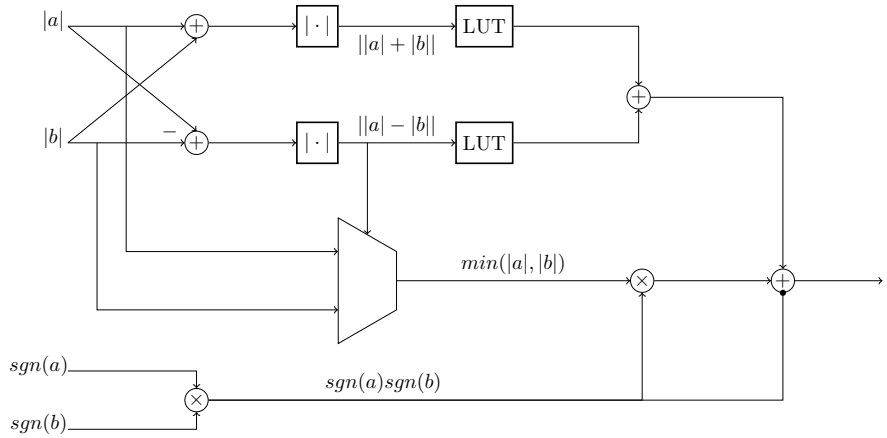


Figure 32: $g(\cdot)$ operator logical scheme.

In order to reduce the performance loss of Min-Sum with respect to Sum-Prod, it is applied the Min-Sum-Offset (γ):

$$\beta_{ji}^{(n+1)} = -\prod_{l=1, l\neq i}^{d_{c_j}} \text{sign}\left(-\alpha_{lj}^{(n)}\right) \cdot \max\left(\left[\min_{l=1, l\neq i}^{d_{c_j}}\left(|\alpha_{lj}^{(n)}|\right) - \gamma\right], 0\right) \tag{21}$$

Based on the substitution:

$$\min(|a|, |b|) \rightarrow \max(\min(|a|, |b|) - \gamma, 0) = \begin{cases} 0 & \text{if} \min(|a|, |b|) < \gamma \\ \min(|a|, |b|) - \gamma & \text{if} \min(|a|, |b|) \geq \gamma \end{cases} \tag{22}$$

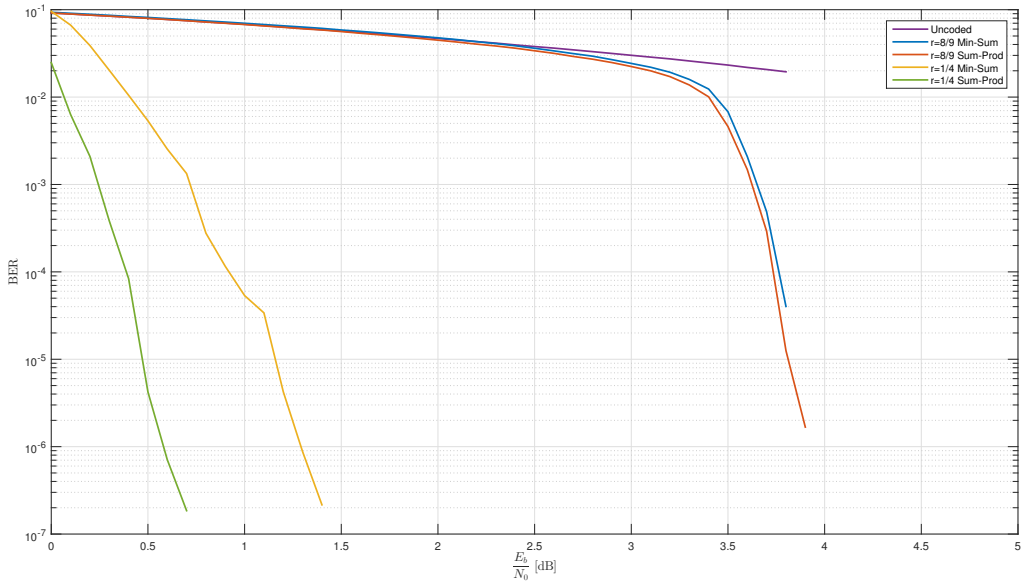
The belief propagation algorithm works as follows:

- At the first step, the check nodes' sockets $\alpha_{ij}^{(0)}$ are initialized to a priori LLRs on coded bits coming from the soft demodulator.
- At each step n,
  - · the variable nodes' sockets $\beta_{ji}^{(n)}$, are obtained either from optimal (17) or suboptimal (21) computation.
  - · the check nodes' sockets $\alpha_{ij}^{(n)}$ are updated according to (16).
- At the end of the iterations, a posteriori LLRs are computed.

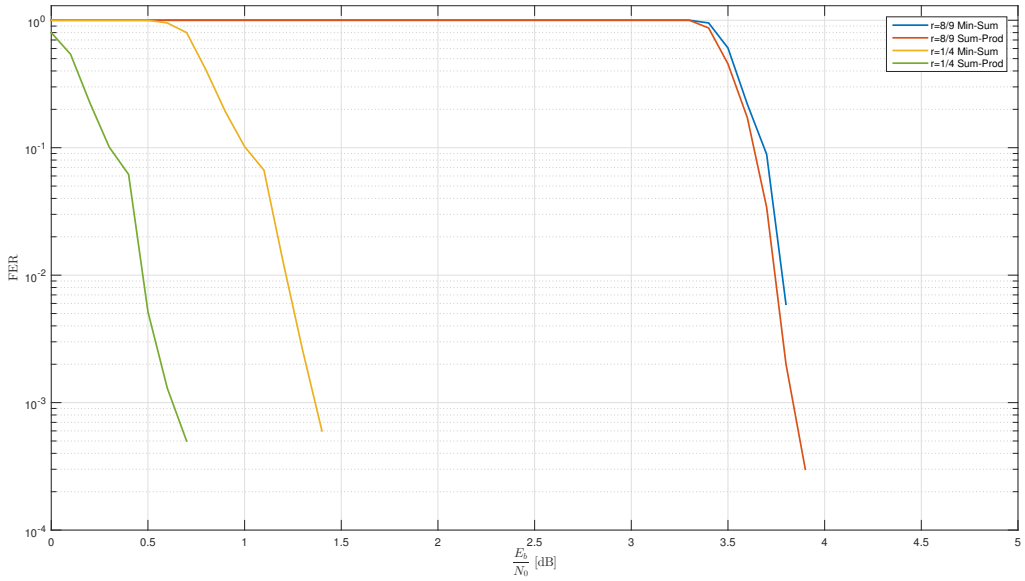
Once the a posteriori LLRs are available, hard decision is performed.

Results Discussion

We simulate the LDPC encoding and analysed the performances by using DVB-S2 Standard.
The Block sizes provided are: short (N = 4096), medium (N = 16200), long (N = 64800).
The possible rates used in DVBS-2 are $r = [1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10]$.
In Figure 33 we plot the BER curves (33a) and the FER curves (33b) using the rates 1/4 and 8/9, with medium block size. We compare the performances of optimal Sum-Prod with respect to Min-Sum with offset $\gamma = 3$. As we expect, using a higher value of rate ($r = 8/9$) does not provide many improvements due to the few amount of added redundancy. Using more redundant codes, $r = 1/4$, we reach an high coding gain. As previously said, using the Min-Sum introduces a loss with respect to the optimal values obtained with Sum-Prod. The loss is roughly 0.5 dB for low E_b/N_0 and it decreases for high E_b/N_0 since the approximation of $\tanh(\cdot)$ with the $\text{sign}(\cdot)$ function is better.
Then we analyse the relationship between the BER and Offset values using Min-Sum-Offset (γ), with rate $r = 1/2$ and short block size.
As we can see in Figure 34, increasing the value of γ, performances improve until we reach an optimal value $\gamma = 4$, then they get worse for higher values.
The optimal value of γ depends on the adopted rate r. For the Optimal value of γ the loss with respect to the Sum-Prod algorithm is roughly 0.1 dB.
Finally, we analyse the performances for the 3 different block length sizes, using the Min-Sum with the optimal value $\gamma = 4$.
In Figure 35 we can see that when the block size is larger, the performances are better. We can notice that the medium length curve crosses the long and the Sum-Prod ones, due to the fact that the DVB-S2 uses a slightly lower rate for this block size ($r = 0.44$ instead of $r = 0.5$).
However, for high E_b/N_0, the long length curve shows a better performance of the medium length one.



(a) Min-Sum and Sum-Prod BER Curves.



(b) Min-Sum and Sum-Prod FER Curves.

Figure 33: Comparison between Min-Sum and Sum-Prod LDPC Performances

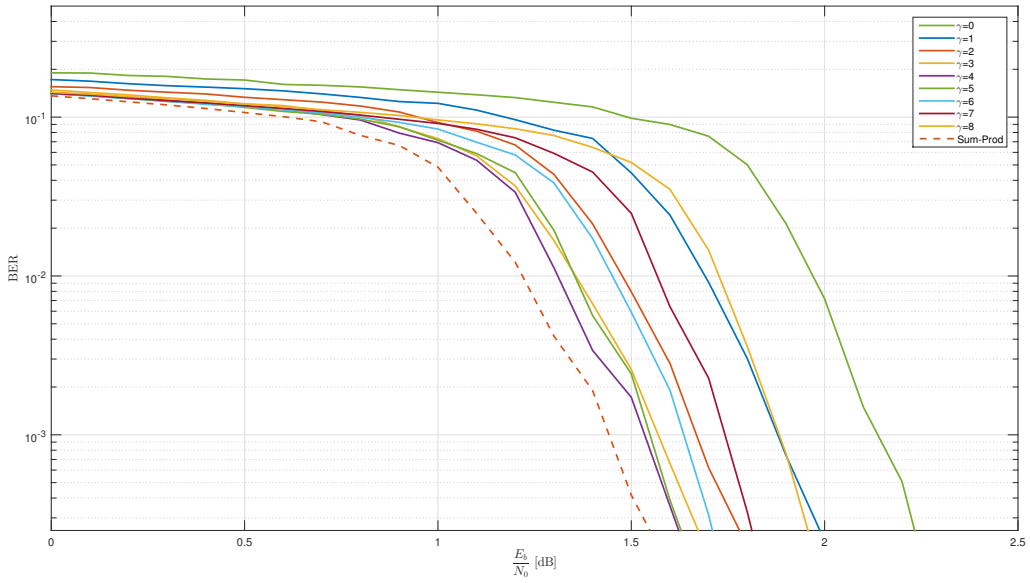


Figure 34: BER Curves of LDPC with $r = 1/2$, short block size, Min-Sum-Offset with $\gamma = [0, 8]$.

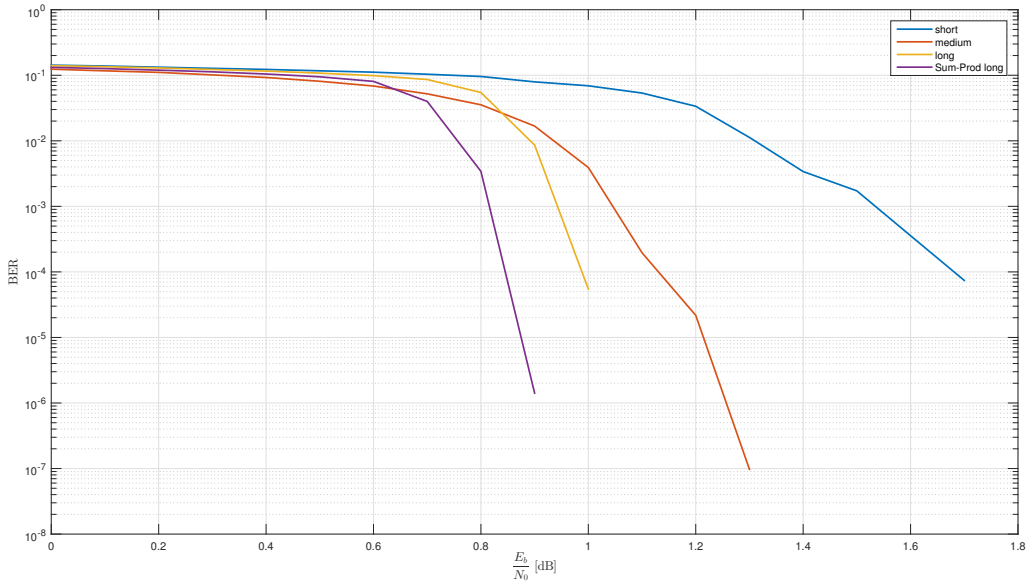


Figure 35: BER Curves of LDPC with $r = 1/2$: short, medium, long block size.