Information Theory and Applications

Lecture notes from the course of prof. Roberto Garello

December 5–6, 2019

Manuele Macchia
Academic year 2019/2020

Contents

1 Discrete systems

1.1 Introduction

A discrete system is a dynamic system that evolves in discrete steps. Therefore, the time axis of a discrete system is discrete: $t = 0, 1, \cdots, i, \cdots$.

$$\begin{tikzcd} {} \arrow[r, "x(i)"] & \boxed{\textsc{System}} \arrow[r, "y(i)"] & {} \end{tikzcd}$$

The input is denoted by $x(i)$. In general the output $y(i)$ depends on:

- the current input $x(i)$
- previous inputs $x(j)$, with $j < i$

1.2 State of the system

The state of the system, denoted with $S(i)$, contains all useful information about the past. When it is known, we do not have to keep track of previous inputs.

Given the state of the system $S(i)$ and the input $x(i)$, we can determine:

- the output $y(i)$
- the next state $S(i+1)$

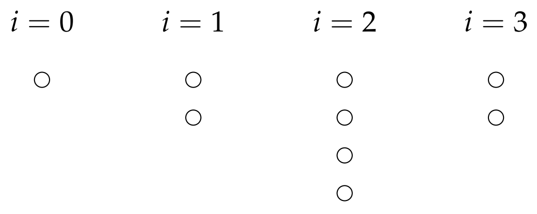
For a general system, $S(i)$ will denote the set of admissible states $s(i)$ at the time $t = i$. We allow $S(i)$ to be different for different time instants.

If the system has n admissible states at the time instant $t = i$, then $S(i)$ will be:

$$S(i) = \{s_1(i), s_2(i), \cdots, s_n(i)\}$$

Note that the number of admissible states of a system can change on the basis of the time instant.

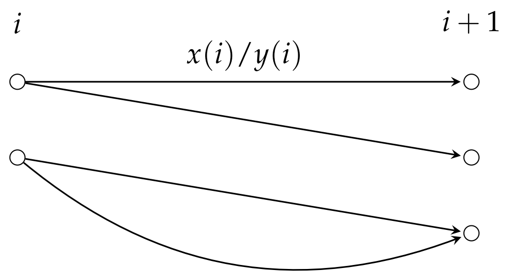
Example 1.1. We can represent the set of admissible states of a system $S(i)$ at each time instant $t = i$ as follows, each circle representing a different state:



1.3 System evolution

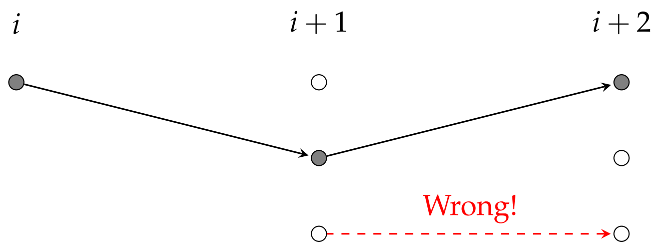
Given a state $s(i)$ at time $t = i$, there are some edges leaving it. Each edge represents the admissible system evolution from that state.

Example 1.2. Consider a system in two consecutive instants of time i and $i+1$. We can represent the system evolution through edges, each edge representing an input $x(i)$ and an output $y(i)$.



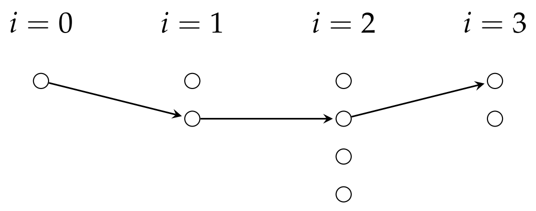
It's important to note that when a system enters a state at time i, it must leave from that same state.

Example 1.3. The system starts from the instant i and follows the trajectory to the second state at the instant $i+1$. The system should now continue its trajectory from this state.



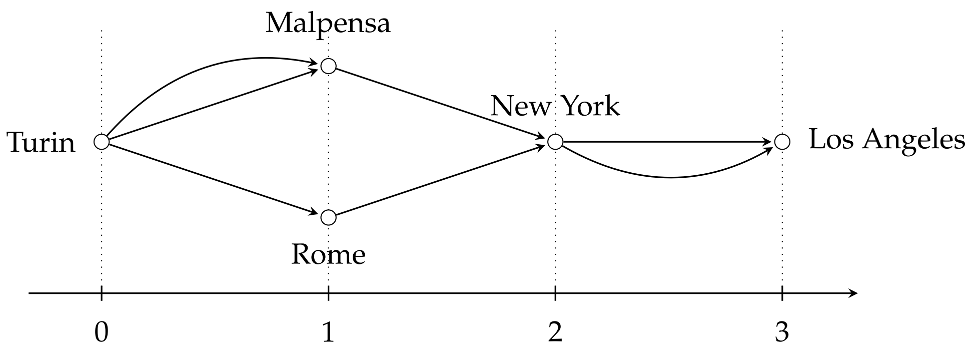
A *trajectory*, or *sequence*, is a sequence of edges from the time instant $i = 0$ to $i = L$. A system is the set of all possible trajectories.

Example 1.4. Consider the example 1.1. We can reach a state in $i = 3$ following this trajectory:



1.4 Example of a discrete system

Let's suppose we want to travel from Turin to Los Angeles. We can consider each discrete instant of time as a step of our trip. Starting from Turin, the only state in $S(i)$, we could either go to Malpensa or Rome, two admissible states of $S(i+1)$. To get to Rome from Turin we can only take a train, but in case we want to depart from Malpensa, we could either take our own car or take a train. These two options are represented as two different trajectories connecting Turin to Malpensa. Any one of these two options can be chosen, but they have different costs. A similar situation is presented to us when we arrive in New York.



1.5 Channel coding

Communication channels model the transmission of information data. An ideal channel reproduces the input at the output, while a real channel output is not always equal to the input. If this is the case, we have a channel error. Channel codes are used to minimize the error probability.

Given a binary information sequence, we can divide it into blocks of k bits, and then transmit $n = k + r$ bits, where r is the number of redundancy bits.

Example 1.5. If we take $k = 2$ and $r = 1$, given the binary sequence

$$0101101011\cdots$$

we can divide it into 2-bits blocks

$$\boxed{01}\boxed{01}\boxed{10}\boxed{10}\boxed{11}\cdots$$

and then transmit the following sequence. Redundancy bits are underlined.

$$\boxed{01\underline{1}}\boxed{01\underline{1}}\boxed{10\underline{1}}\boxed{10\underline{1}}\boxed{11\underline{0}}\cdots$$

1.5.1 Parity code

The redundancy bit(s) can be exploited in order to detect errors upon receiving data. An example of channel code is the parity code, which computes a redundancy bit (called parity bit) so that each block of bits with length $n = k + r$ always has an even number of ones. This property can then be checked by the receiving end of the channel in order to detect errors in trasmission.

Suppose that we choose $k = 2$. Each block can be treated as an input vector $\boldsymbol{u} = (u_1, u_2)$. The parity bit adds one bit to each block, so that $n = k + 1 = 3$. Therefore, another vector $\boldsymbol{c} = (c_1, c_2, c_3)$ which contains the parity bit can be computed as follows.

$+$	0	1
0	0	1
1	1	0

- $c_1 = u_1$
- $c_2 = u_2$
- $c_3 = u_1 + u_2$

We can list all possible inputs and outputs.

$$\begin{aligned}
\boldsymbol{u} = (0,0) &\quad\rightarrow\quad \boldsymbol{c} = (0,0,0)\\
\boldsymbol{u} = (1,0) &\quad\rightarrow\quad \boldsymbol{c} = (1,0,1)\\
\boldsymbol{u} = (0,1) &\quad\rightarrow\quad \boldsymbol{c} = (0,1,1)\\
\boldsymbol{u} = (1,1) &\quad\rightarrow\quad \boldsymbol{c} = (1,1,0)
\end{aligned}$$

1.5.2 Repetition code

A more powerful channel code can be exploited not only for error detection, but for error correction as well. To achieve this, the idea of the repetition code is to just repeat the message several times.

Suppose that we choose $k = 1$, so that each information bit is treated as a separate block, and $r = 2$, so we repeat the message two more times, therefore $n = 3$. We can have the following inputs and outputs, with redundancy bits underlined.

$$\begin{aligned}
0 &\quad\rightarrow\quad 0\underline{00}\\
1 &\quad\rightarrow\quad 1\underline{11}
\end{aligned}$$

Suppose that we transmit 000 over a communication channel, and the other end receives 010. This is an incorrect transmission, although we can recover the original information. We can compute the Hamming distance (the number of different bits) between the message received and the only two valid messages (000 or 111).

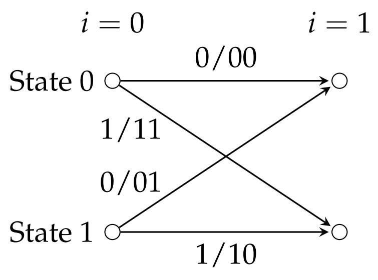
$$\begin{aligned}
010 \; vs \; 000 &\quad\rightarrow\quad d_H = 1\\
010 \; vs \; 111 &\quad\rightarrow\quad d_H = 2
\end{aligned}$$

If we take the closest code, we get 000, which is the original message sent over the channel. We have corrected the error.

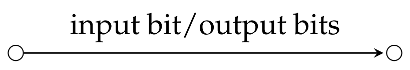
1.5.3 Convolutional code

Convolutional codes are powerful channel codes. They are described by a state diagram called *trellis*.

Example 1.6. The following is an example of a trellis diagram.



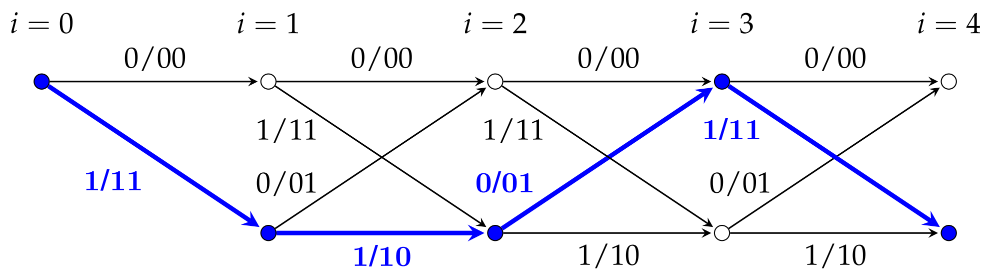
As we can see, each edge depends on the input bits and the output bits.



Example 1.7. Suppose that we want to encode the following bits ($k = 1$ and $n = 2$, thus $r = 1$).

1	1	0	1

The following figure represents the trajectory of the system.



The system gives then the following output.

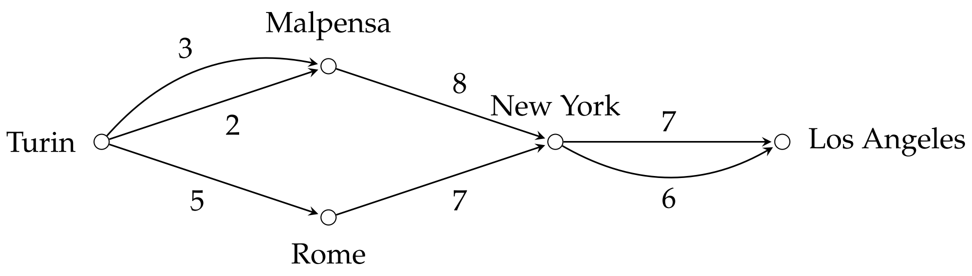
11	10	01	11

2 Viterbi algorithm

2.1 Cost of trajectories

Suppose that each trajectory has a cost. The cost of a trajectory is defined as the sum of the cost of the individual edges.

Let's consider the example presented in section 1.4. Suppose each trajectory has a cost that represents the hours needed to get to the next city.

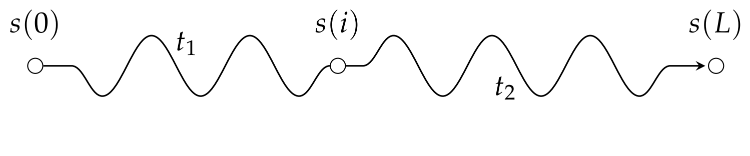


The problem to solve is finding the minimum cost trajectory. This can always be solved through an exhaustive brute force approach. First, list all possible trajectories. Then for each trajectory compute the cost, which is the sum of the cost of its edges. Finally, select the minimum cost trajectory.

Consider now the example 1.7. There are 2^4 different trajectories that can be taken to reach the end. In this case, a brute force approach is feasible. Although, when L grows, the number of possible trajectories 2^L grows exponentially, and the brute force approach quickly becomes too computationally expensive. If so, the Viterbi algorithm can be used to solve this minimization problem.

2.2 Bellman theorem

The Viterbi algorithm is based on the Bellman theorem. Suppose that the following is the minimum cost trajectory between $s(0)$ and $s(L)$, and that it passes through $s(i)$ so that the trajectory is split in two parts t_1 and t_2.



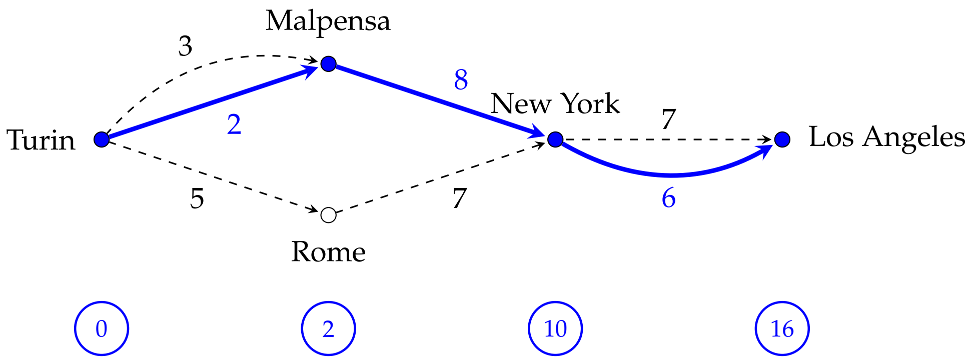
Theorem t_1 is the minimum cost trajectory between $s(0)$ and $s(i)$.

Proof If t_1' is a trajectory between $s(0)$ and $s(i)$ with cost less than t_1, the minimum cost trajectory from $s(0)$ to $s(L)$ passes through t_1'.

2.3 Viterbi algorithm

The idea behind the Viterbi algorithm is the following. Starting from the time instant $i = 0$, we move left to right. At any time i, for any state $s(i)$, we only keep the minimum cost trajectory entering it. All the other ones can be discarded, because they will never be part of the winning trajectory.

Consider the example presented in section 1.4. The trajectory to keep is highlighted in bold. The weight of the winning trajectory (16) is called *state metric*. The weight of each edge is called *edge metric*.

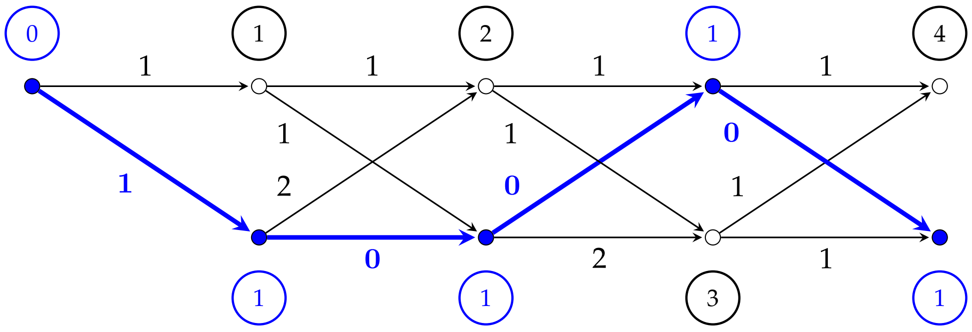


Now consider example 1.7. The system produces the sequence of bits 11 10 01 11. Due to noise over the communication channel, the sequence received on the other end differs from the one transmitted: $1\underline{0}$ 10 01 11.

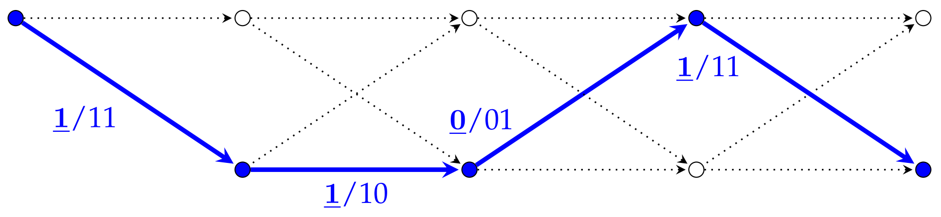
Suppose we want to correct errors introduced by the communication channel and recover the transmitted sequence of bits. We can solve this problem through a brute force exhaustive approach. To do so, we can list all L different trajectories, then compute the Hamming distance d_H between each trajectory and the received sequence of bits, and finally select the minimum distance trajectory.

$$\begin{array}{cccccccc} & & & & & & d_H \\ 0000 & \rightarrow & \underline{0}0 & \underline{0}0 & 0\underline{0} & \underline{00} & 5 \\ 1000 & \rightarrow & 1\underline{1} & \underline{01} & 0\underline{0} & \underline{00} & 6 \\ \vdots & & & & & & \\ 1101 & \rightarrow & 1\underline{1} & 10 & 01 & 11 & \textcircled{1} \\ \vdots & & & & & & \\ 1111 & \rightarrow & 1\underline{1} & 10 & \underline{10} & 1\underline{0} & 4 \end{array}$$

This process can be computationally expensive, therefore it is only feasible when L is not too big. Another way to solve this problem is to apply the Viterbi algorithm. The cost of each edge of the diagram is the Hamming distance between the received block of bits and the output bits.



The lowest cost trajectory is highlighted in blue. When we move left to right and select the edge with the lowest cost, we can store the input bit at each step. At the end of the algorithm, we'll have the original sequence of bits, as shown in the following diagram.

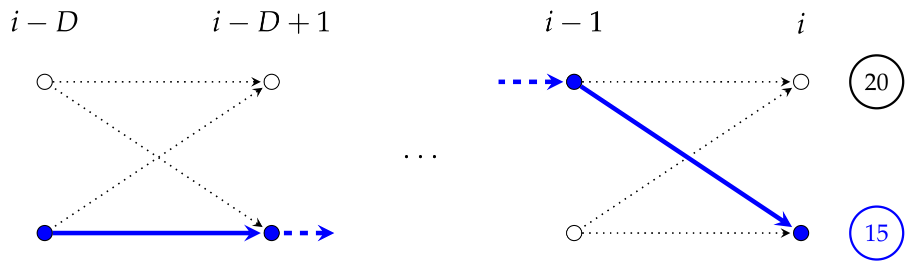


2.4 Early decision Viterbi algorithm

In the original Viterbi algorithm we must wait until the end of the sequence before we can make a decision on the winning trajectory and release the corresponding information bits. If the sequence is a very long string of bits, we introduce latency.

It's possible to overcome this problem using a modified version of the Viterbi algorithm, called early decision Viterbi algorithm. This algorithm takes a decision on the information to release before the end of the sequence.

Suppose we're processing section i. We identify the minimum cost trajectory in section i. In section $i - D$, it passes through a given edge. We release the information bit labeling this edge. This way, we release the information bits with a constant latency D, without waiting until the end of the sequence.

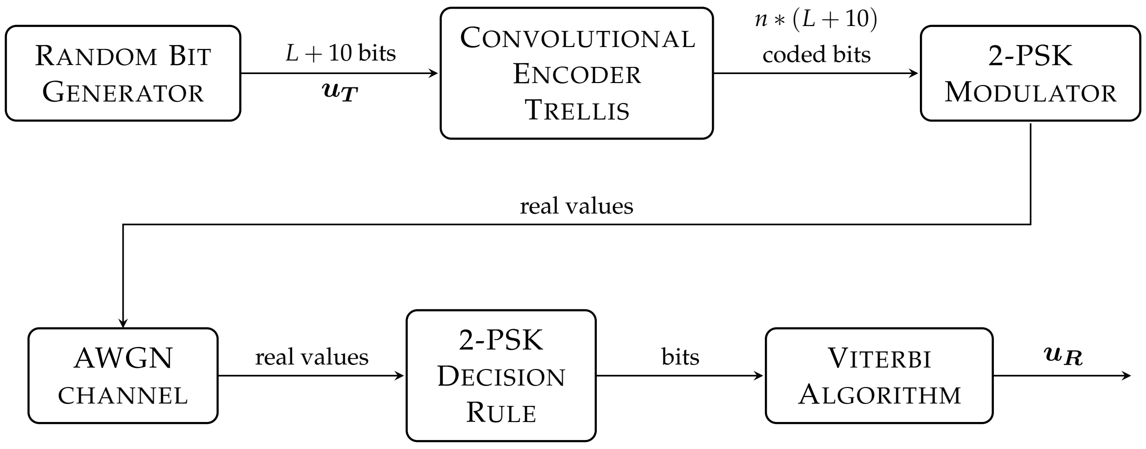


This algorithm is not optimal, but it reduces latency. At the receiver side we release the information bits with a constant small latency D, instead of waiting until the end of the entire sequence.

This algorithm is suboptimal because the best trajectory at time i may not be part of the minimum cost trajectory at the end of the sequence. On the other hand, it can be observed that if D is chosen big enough, the loss with respect to the optimal Viterbi algorithm is negligible.

3 Matlab program

3.1 System diagram

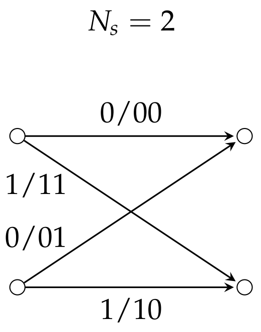


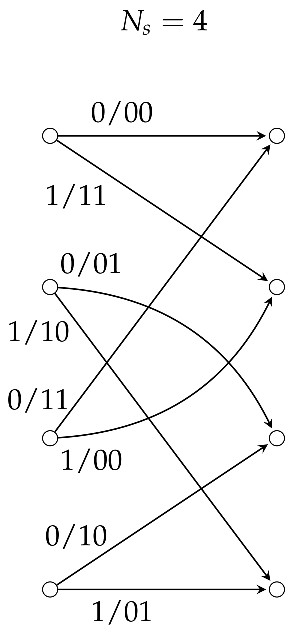
3.2 Random bit generator

Generate a sequence of L random bits. Add 10 zero bits at the end of the sequence, so that the sequence passed to the convolutional encoder is $L + 10$ bits long.

3.3 Convolutional encoder trellis

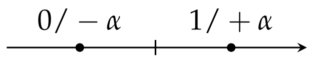
Use the following trellis diagrams to test the algorithm. The number of states N_s should be a parameter.





3.4 2-PSK modulator

Transmit $-\alpha$ or $+\alpha$ depending on the input bit. It's suggested to choose $\alpha = 1$.

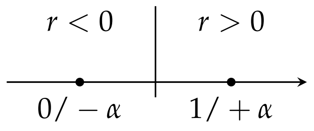


3.5 Additive white gaussian noise (AWGN) channel

Add to each symbol $-\alpha$ or $+\alpha$ a sample of a gaussian random variable with zero mean and variance $\sigma^2 = N_0/2$. For example, if we want to transmit $+\alpha = 1$ with noise -0.1, the symbol received will be 0.9.

3.6 2-PSK decision rule

Decide whether the symbol received corresponds to a 0 or a 1. The symbol received is denoted by r. Due to gaussian noise, some of the bits may differ from the ones transmitted originally.

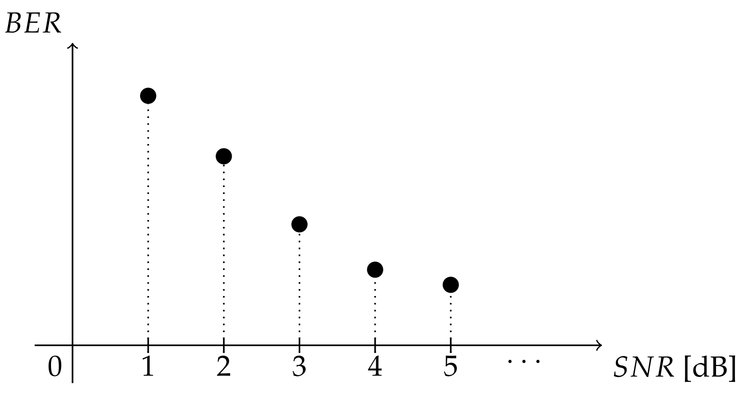


3.7 Viterbi algorithm

Apply the Viterbi algorithm to the $n * (L + 10)$ received bits. This block outputs a string of $L + 10$ bits.

3.8 Output

At the end, compare $\boldsymbol{u_T}$ and $\boldsymbol{u_R}$ to compute the number of wrong bits. The program should display a graph plotting the signal to noise ratio (SNR) on the X axis and the bit error rate (BER) on the Y axis.



The bit error rate can be computed as follows. Remember that the last 10 bits (zeroes) do not contain any information, and therefore should not be considered in the computation.

$$BER = \frac{\textit{\# wrong bits received}}{\textit{\# bits received}}$$

The signal to noise ratio is expressed in decibels. Remember that $x\ dB = 10 \log_{10} x$.

$$SNR = \frac{energy\ of\ the\ signal}{energy\ of\ the\ noise} = \frac{\alpha^2}{N_0}$$

Consider 1 dB increments ($SNR = 0,\ SNR = 1,\ \cdots$) and for each one generate enough information bits until the BER is lower than 10^{-5}. We expect that when SNR increases, BER decreases.

$$SNR = 0\ dB \implies SNR = 1 \quad \longrightarrow \quad \sigma^2 = \frac{1}{SNR} = 1$$