Some notes for the course

INFORMATION AND CODING

Armando J. Pinho

***Important note**: This document contains a summary of some of the topics addressed in the course "Information and Coding". As such, it should be regarded as a guide and a complement, and by no means the only source of information nor a substitute of the classes.*

(Draft version of October 17, 2021)

Contents

1 Introduction

What is "**Information**"?

In this section, there are some questions that, in some way, have to do with the way we may define and measure information. Although it is not expected that you have answers for all of them right now, you should nevertheless start thinking about them.

Discussion topic 1.1
Consider the following questions and try to sketch answers for them:

- *What is data?*
- *How to measure the quantity of data?*
- *What is information?*
- *How to measure the quantity of information?*
- *What is a computation?*
- *Are there problems that computers cannot solve?*

Discussion topic 1.2
Consider the following statement:

> *One bit of **data** contains, at most, one bit of **information**.*

Do you think this is true? Why?

Discussion topic 1.3
*Alice told Bob that she found an image compression software that is able to reduce the size of **every** image in, at least, 50% of its original size—for example, if originally the image occupies* $1\,000\,000$ *bytes, after compression it will require at most* $500\,000$ *bytes.*

*Bob was not very impressed by Alice's statement. However, when Alice added that she would also be able to **always** recover the original image from its compressed version, Bob immediately replied:*

> *"THAT IS IMPOSSIBLE! Every lossless (i.e., reversible) compression method is limited, i.e., it cannot compress all messages!"*

Is Bob correct? Why?

Discussion topic 1.4

Consider the following game, which is a case of a Post canonical system[1]*. The objective of the game is to discover the sequence of rules (among a previously specified set) that allows transforming a certain word into another word.*

Consider, for example, the case where the words can be formed using only the letters M, I and U, and that the allowed transformation rules are:

1. $xI \rightarrow xIU$
2. $Mx \rightarrow Mxx$
3. $xIIIy \rightarrow xUy$
4. $xUUy \rightarrow xy$

Let us try some transformations:

1. *Give the steps to transform MI into MUI.*
2. *Can you transform UIM into MIU? Why?*
3. *And MIU into UIM?*
4. *What are the steps to transform MI into MU?*

Can you think of an algorithm that answers questions of the type "Is it possible to transform the word w into the word z, using a certain set of rules"?

Discussion topic 1.5

Consider the set of all functions $f : \mathbb{N} \rightarrow \{0, 1\}$. Alice told Bob that there are functions in that set that cannot be calculated by any finite program, regardless of the programming language used to implement it. Bob thinks Alice is wrong. Is she wrong? Why?

Discussion topic 1.6

Alice wants to play a game with Bob, where a coin has to be tossed. To show that she is using a fair coin, she tosses it twenty times. However, in all trials, heads comes up. Can Bob trust this coin? Why?

Would it be different if the sequence of outcomes had been "00101001110101010110"? Why?

[1] This problem can be reformulated as a string rewriting system, also known as a semi-Thue system, which is related to the notion of unrestricted grammar.

Discussion topic 1.7
Consider the following procedure, suggested by John von Neumann (1903–1957):

1. *Toss a coin twice*
2. *If the results match, start over, forgetting both results*
3. *If the results differ, use the first result, forgetting the second,*

with which he claimed to be possible to obtain the equivalent to a sequence generated by a fair coin, even if the coin is unfair.

Do you agree with him? Why?

Discussion topic 1.8
You work at a company that sells large sequences of random numbers. Your boss thinks he needs to have some way of testing the "randomness" of the sequences before they are sent to the clients (let us consider this as a kind of quality control of the production...). Because you are the best informatics engineer of the company, he asks you to develop a program for testing the randomness of arbitrary sequences.

What should you answer him? Have you any ideas of how to do it? Can you define "randomness"?

Discussion topic 1.9
John von Neumann once said:

> *"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."*

What do you think he meant to say with this statement?

Discussion topic 1.10
Consider the following definition of a certain number:

"The smallest positive integer not definable in fewer than twelve words".

Does this number exit? Why?

(This is known as Berry's paradox.)

Discussion topic 1.11
You are at a party with friends. After some drinks, one of them says:

> *"I've had a great idea! I'll write a program that will be capable of analysing other programs and tell us if they are bug-free or not! I wonder why no one have done this before!"*

What would you say to your friend?

Discussion topic 1.12
Consider the following segment of pseudo-code:

```
F(uint x) {
   while(x > 1) {
      if(x % 2 == 0)
         x = x / 2
      else
         x = 3 * x + 1
   }

   return
}
```

Does function F return for all x?

(This is known as Collatz's conjecture.)

---

2 A combinatorial measure of information

Problem 2.1
Alice is tossing a coin repeatedly. She wants to send the outcomes to Bob. To send the ***message****, Alice is only able to use a flashlight, by turning it on and off once per second.*

1. *Suppose that Alice and Bob agreed on the size of the message, n, beforehand, i.e., on how many coin tossing outcomes will be sent in each message. Propose a procedure for this transmission scheme.*
2. *Suppose now that Alice intends to send several messages of lengths, n_1, n_2, ..., not known in advance by Bob. Can you suggest a transmission procedure that works in this case?*

Problem 2.2
Alice wants to send a letter to Bob, using the same transmission scheme. Consider an ***alphabet*** *of 27* ***symbols*** *(26 letters plus the space). How can this be done?*

Problem 2.3
Now Alice wants to play a game with Bob (they are finally face to face!). She thinks of a number between 1 and 100. Bob is allowed to ask every question he wants, but only gets a yes or no answer.

1. *What is the minimum number of questions that Bob needs to sequentially pose to Alice (the next question is posed only after knowing the answer to the previous question), in order to discover the number Alice thought of? What should those questions be?*
2. *Now Bob is only allowed to pose all the questions at once. Is the minimum number of required questions the same? In this case, what should be the questions?*

The minimum number of sequential questions is attained, for example, by successively halving the set of possibilities (a kind of binary search...). Taking as example the game of guessing the number, we have

$$2^b = 100,$$

where b is the number of questions. Hence,

$$b = \log_2 100 \approx 6.644,$$

meaning that, on average, about 6.644 questions are needed to find the number—in some cases six questions are enough, whereas in other cases seven questions are needed.

Example 2.1
Guess a number between 1 *and* 100*:*

$$\begin{gathered}\textit{Is } x \geq 50? \rightarrow 1\\ \textit{Is } x \geq 75? \rightarrow 0\\ \textit{Is } x \geq 62? \rightarrow 1\\ \textit{Is } x \geq 68? \rightarrow 0\\ \textit{Is } x \geq 65? \rightarrow 0\\ \textit{Is } x \geq 63? \rightarrow 1\\ \textit{Is } x = 63? \rightarrow 0\end{gathered}$$

Hence, x *has to be* 64 *(required seven yes/no questions).*

Example 2.2
Guess a number between 1 *and* 100*:*

$$\begin{gathered}\textit{Is } x \geq 50? \rightarrow 1\\ \textit{Is } x \geq 75? \rightarrow 0\\ \textit{Is } x \geq 62? \rightarrow 1\\ \textit{Is } x \geq 68? \rightarrow 0\\ \textit{Is } x \geq 65? \rightarrow 1\\ \textit{Is } x \geq 66? \rightarrow 0\end{gathered}$$

Hence, x *has to be* 65 *(in this case, six yes/no questions were enough).*

Problem 2.4
Using this searching scheme, the average number of questions needed is 6.72*. Verify this value (by calculating how many of the 100 numbers require the seven questions to be defined). Why is this value larger than the theoretical bound of* $\log_2 100 \approx 6.644$ *bits?*

Now imagine that x *is represented in binary. What should be the "natural" questions to pose in this case? Using this approach, can you attain an average number of questions smaller than* 6.72*?*

Notice that, if all the questions had to be provided at once, then in the example of guessing a number between 1 and 100 we had to ask always a minimum of $\lceil \log_2 100 \rceil = 7$ questions.

Problem 2.5
Suppose now that the number to be guessed is between 1 *and* 3*. Compare and discuss the impact on the average number of questions that have to be posed if:*

(a) Questions are posed sequentially, after knowing the previous answers;

(b) The questions are provided in batch.

Discuss also aspects such as the case where the numbers are not equally probable and also the difference between $\log_2 3 \approx 1.585$ *and the average number of questions estimated for the several scenarios addressed.*

Programming 2.1
Write a program that generates the minimal sequence of yes/no answers that allows discovering a certain number between 1 *and* n*.*

Problem 2.6
Using the ideas above, show that, in general, sorting has an average case lower bound of $\Omega(n \log n)$ *comparison operations.*

Problem 2.7
Because the game was getting boring, Alice says to Bob that she might start lying in, at most, one of the yes/no questions she is answering. Can you help Bob finding out the minimum number of additional questions he needs to pose in order to compensate for a possible wrong answer of Alice?

In order to be able to correct a possible error, we may use the simple scheme of posing the same question three times and then decide by majority (in this case, we should use a total of 21 questions). This approach is known as a **repeat code**. If the questions are posed sequentially, it is easy to see that this number can be reduced, in the worst case, to 15 (How?). Moreover, even if all questions are posed at once, it is possible to solve the problem with only 11 questions!

To see how this can be done, we have first to be convinced that one-bit error correction is possible if each **codeword** differs at least in three bits from all other possible codewords (i.e., the **Hamming distance** between codewords is at least three). So, each codeword needs to have a "forbiden region" around it of radius one (known as a radius-one ball):

- A n-bit string has $n + 1$ strings at Hamming distance less or equal than one.
- For k bits of data, we need 2^k codewords and, hence, 2^k disjoint balls.
- This requires $2^k(n + 1)$ distinct n-bit strings.
- Therefore,

$$2^n \geq 2^k(n + 1), \quad \text{and} \quad k \leq n - \log_2(n + 1).$$

In the case of our problem ($k = 7$), this condition is satisfied for $n = 11$. An effective code can be constructed using ideas initially proposed by Richard Hamming (1915–1998) (Hamming, 1950). Although without entering into much detail, we can see the **Hamming codes** as build by introducing parity bits (computed according to specific rules) in the positions of the codeword that are powers of two. In our case, the codewords would have the form

$$\boxed{p_1|p_2|d_1|p_3|d_2|d_3|d_4|p_4|d_5|d_6|d_7}$$

where the p_i are the parity bits and the d_i the data bits[2], originating a code with a rate of $7/11 \approx 0.64$.

Problem 2.8
Three players wearing hats enter a room. The hats can be black or white and are assigned by flipping a fair coin. The rules of the game are:

- *No player can see his own hat, but can see the hat of every other player.*
- *When in the room, the players cannot communicate. However, before having the hats, they are allowed to decide on a game strategy.*
- *Each player can either announce his guess regarding his hat color or pass.*
- *All players will do the announcement or pass simultaneously.*

The group wins the game if at least one person guesses correctly and no one guesses incorrectly.

What is the best strategy to give the group the highest possible probability of winning the game? Can you relate this problem with a Hamming code of length three?

With $n = 3$, we can form $2^3 = 8$ different 3-bit strings. Suppose that we pick the strings 000 and 111 as the possible codewords. Then, the set of 8 3-bit strings form a Hamming$(3, 1)$ code (verify this). If each of the 3-bit strings occurs with equal probability, then there is a $1/4$ probability of generating a correct codeword (2 out of 8) and a $3/4$ probability of generating a erroneous codeword (6 out of 8). Therefore, the strategy for the game is the following:

1. If the player sees the other two players with the same hat color, then he assumes that he has a different hat color, because this would be the most probable case. Hence, he announces that color.

[2]Of course, the position of the parity bits is arbitrary, provided that both the encoder and decoder know their positions. For example, the Hamming$(7, 4)$ can be obtained using

$$d_1, d_2, d_3, d_4, d_2 \oplus d_3 \oplus d_4, d_1 \oplus d_3 \oplus d_4, d_1 \oplus d_2 \oplus d_4.$$

2. If the player sees the other two players with different hat colors, then, since in that case the probability of his hat having any of the two colors is the same, he passes.

Notice that, using this strategy, the probability of choosing the wrong color is still 50% each time someone announces a color. The gain is that the wrong guesses are not evenly distributed—from the total of twelve announcements associated to the eight combinations, the six that are wrong are concentrated in just two cases; all other six cases are error free.

Homework 2.1
Try to find out what is the highest probability of winning the game when there are seven players. Hint: use the Hamming$(7, 4)$ *code. What should be the game strategy in this case?*

The problems that we have been addressing consider a measure of information that is usually called **combinatorial**, because it is only concerned with the number of possible objects involved, assuming that they occur with equal probability. Hence, for m distinct objects (a size-m alphabet), the amount of **combinatorial information** needed to specify each one is

$$\log_2 m \quad \text{bits.}$$

This approach for measuring information can be traced back at least to the works of Nyquist and Hartley:

- Harry Nyquist (1889–1976), *Certain Factors Affecting Telegraph Speed*, Bell System Technical Journal (Nyquist, 1924):

  > *"This paper considers two fundamental factors entering into the maximum speed of* ***transmission of intelligence*** *by telegraph. These factors are signal shaping and choice of codes."*

- Ralph Hartley (1888–1970), *Transmission of Information*, Bell System Technical Journal (Hartley, 1928):

  > *"A quantitative measure of "information" is developed which is based on physical as contrasted with psychological considerations."*

In his paper, Hartley proposes measuring the information content, H, of a message as

$$H = n \log m = \log m^n,$$

where m denotes the number of possible symbols (the size of the alphabet) and n the number of symbols in the message. So, Hartley defined the amount of information in a message as the logarithm of the number of possible messages (message space).

The base of the logarithm determines the unit of information used: **hartley** for base 10, **nat** for base e, **bit** for base 2.

Problem 2.9

Alice has a deck of playing cards (52 *cards). After shuffling the deck, Alice wants to send enough information to Bob in order for him to put his deck in the same order. How many bits does Alice need to send to Bob to communicate this information? Can you propose a method for doing it?*

3 A probabilistic measure of information

Let us return to Alice and Bob and consider the following new problem:

Problem 3.1
Using the flashlight communication system, Alice wants to communicate to Bob the results of throwing a pair of dice (i.e., a number from 2 to 12). However, Alice is running out of batteries for the flashlight and, therefore, she wants to preserve them as much as possible. So, she wonders if, in this case, it is possible to use, on average, less than $\log_2 11 \approx 3.46$ *bits for sending to Bob the outcome of each trial... Can you help her solving this problem?*

The first aspect to take into consideration is that the possible outcomes of rolling two dice do not occur with the same frequency. Let us denote by Σ the alphabet representing those outcomes, i.e., $\Sigma = \{2, 3, \ldots, 12\}$. For example, there is only one combination for getting a twelve, whereas for getting a seven there are six possible combinations. Hence, Alice will have to transmit symbol "7" about six times more often than the symbol "12" of the alphabet.

The key idea is to use shorter representations, i.e., less bits, for the most frequent symbols.

The Morse code (19th century) is a good example of this principle, recognized well before the foundations of the theory of information were established by Claude Shannon (1916–2001) in the mid of the 20th century:

- Claude Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal (Shannon, 1948):

  > *"The recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist and Hartley on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the* ***statistical structure of the original message*** *and due to the nature of the final destination of the information."*

Discussion topic 3.1
The Morse code is composed of sequences of dots and dashes of various lengths, according to the convention shown in Fig. 3.1. In this code, the characters of a word need to be separated by additional space (three units of time, where one unit of time is equivalente to the length of a dot). For example, the letters "ET" originate the sequence "· —", to distinguish from the letter "A" that is encoded as "· —". Do you think this explicit separation could be avoided? If you think so, how should the code be constructed?

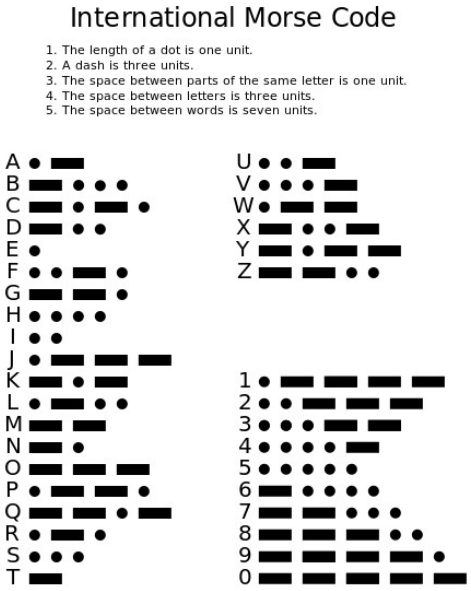


Figure 3.1: Morse code table.

3.1 About codes

Consider an alphabet with m symbols, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, and the corresponding **estimated frequencies of occurrence** of each symbol, p_i. Of course,

$$\sum_{i=1}^{m} p_i = 1.$$

If symbol $\sigma_i \in \Sigma$ is represented by a codeword w_i, then the objective is to **minimize**

$$\sum_{i=1}^{m} p_i l_i, \tag{3.1}$$

where $l_i \in \mathbb{N}^+$ is the length of the codeword w_i, i.e., its number of bits. Often, we will represent the length of a binary string w_i by $|w_i|$, i.e., $l_i = |w_i|$.

Obviously, we need to impose additional constraints to the minimization of (3.1), otherwise we could arrive at completely useless solutions, such as $l_i = 1, \forall_i \ldots$

Discussion topic 3.2
Can you suggest restrictions that should be imposed in the minimization of (3.1)?

Of course, one of the restrictions should be that $w_i \neq w_j, \forall_{i \neq j}$, otherwise the code is not reversible, i.e., it is not possible to recover the original message. But this condition is not

enough for reversibility (can you see why?). In fact, we need a stronger condition: The code should be **uniquely decodable**. This implies that two different messages will always have two different encodings. Recall the encoding of "ET" and "A" using the Morse code of Fig. 3.1 and the need for additional space to separate the letters. Without it, two different messages would originate the same encoding, i.e., "· —".

Note that, for our purposes, an **encoding** is the assignment of binary strings to elements of the alphabet. Some encodings are of **fixed length** (the ASCII encoding is an obvious example), whereas other encodings are of **variable length**. To attain compression, we need the latter ones.

Problem 3.2
Find codes for which $w_i \neq w_j, \forall_{i \neq j}$, but that are not uniquely decodable.

A sufficient (but not necessary) condition for unique decodability is **immediate decodability**. This means that symbol σ_i is immediately determined as soon as the last (i.e., the rightmost) bit of w_i is read during decoding. For this to happen, the codewords need to be **prefix-free**, i.e., a shorter codeword cannot be a prefix of a longer codeword (why do we need this requirement?).

Problem 3.3
*Show that the Morse code is not a **prefix-free code** (Suggestion: try to build a decoding tree). What are the practical implications of this limitation of the Morse code?*

Letters	Probability	Code 1	Code 2	Code 3	Code 4
a_1	0.5	0	0	0	0
a_2	0.25	0	1	10	01
a_3	0.125	1	00	110	011
a_4	0.125	10	11	111	0111
Average length		1.125	1.25	1.75	1.875

(Sayood 2012)

Table 1: Example of several codes, with different properties.

Table 1 shows four different codes for a 4-symbol alphabet (although not all of them are useful...). They have different properties, namely:

- Code 1 is not uniquely decodable.
- Code 2 is also not uniquely decodable.
- Code 3 is uniquely decodable and instantaneous.

- Code 4 is uniquely decodable, but not instantaneous.

Problem 3.4
Consider now the following uniquely decodable, but not instantaneous, code:

Symbol	*Codeword*
σ_1	*0*
σ_2	*01*
σ_3	*11*

How to decode the string "01111111111111111"?
(Note that the first symbol can be either σ_1 or σ_2...)

Problem 3.5
Consider now the following code:

Symbol	*Codeword*
σ_1	*0*
σ_2	*01*
σ_3	*10*

How to decode the string "01010101010101010"?
(This code is neither instantaneous nor uniquely decodable)

Algorithm 3.1 (Unique decodability)
Suppose that we have two binary codewords a and b, with $k = |a|$, $n = |b|$, with $k < n$, and where a is a prefix of b. We refer to the last $n - k$ bits of b as the dangling suffix.

The algorithm for finding if a code is uniquely decodable is the following:

- *Construct a set with all the codewords.*
- *For every pair of codewords, generate the dangling suffix (if it exists) and add it to the set.*
- *Repeat until:*
  (a) You get a dangling suffix that is a codeword;
  (b) There are no more unique dangling suffixes.

If the algorithm terminates with condition (a), i.e., if it finds a dangling suffix that is a codeword, then the code is not uniquely decodable.

Example: $\{0, 01, 10\} \rightarrow \{0, 01, 10, 1\} \rightarrow \{0, 01, 10, 1, 0\}$.

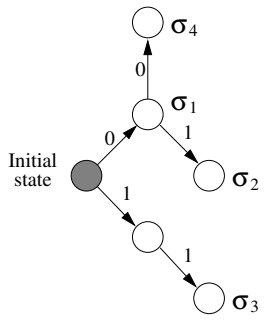


Figure 3.2: Example of a decoding tree of a non instantaneous (and also non uniquely decodable) code.

If the algorithm terminates with condition (b), i.e., if there are no more unique dangling suffixes, then the code is uniquely decodable.

Example: $\{0, 01, 11\} \rightarrow \{0, 01, 11, 1\}$.

As mentioned, if, in a certain encoding, none of the codewords is a prefix of another codeword, then the code is necessarily uniquely decodable. An easy way of checking if a code is prefix-free is to build the corresponding rooted binary tree. In a prefix-free code, the codewords are only associated with the external nodes of the tree.

Example 3.1
Consider an alphabet with symbols σ_1, σ_2, σ_3 and σ_4, and the following codeword assignment: $\sigma_1 \rightarrow 0, \quad \sigma_2 \rightarrow 01, \quad \sigma_3 \rightarrow 11, \quad \sigma_4 \rightarrow 00$.

If the decoder receives "0001", then it cannot determine if the encoder has sent the string "$\sigma_1\sigma_1\sigma_2$" or "$\sigma_4\sigma_2$". As can be seen using the corresponding decoding tree (Fig. 3.2), this code has several problems:

- *The path to node σ_3 is done by an intermediate node that does not contain a bifurcation: the code is inefficient.*
- *Node σ_1 is not a terminal node. The paths to nodes σ_2 and σ_4 include node σ_1: the code is not instantaneous (in fact, it is also not uniquely decodable, as can be seen using Algorithm 3.1).*

Example 3.2
Let us now consider that each branch of the tree connects to a terminal node or to a decision node that leads to a terminal node. Based on this restriction, we arrive at the following codeword assignment: $\sigma_1 \rightarrow 0, \quad \sigma_2 \rightarrow 10, \quad \sigma_3 \rightarrow 110, \quad \sigma_4 \rightarrow 111$. *As can be confirmed (see Fig. 3.3), this code is uniquely decodable (and also instantaneous).*

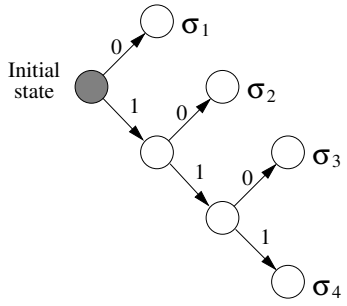


Figure 3.3: Example of a decoding tree of a uniquely decodable (an instantaneous) code. This code is prefix-free.

Another way of verifying if a code is prefix-free is to map the codewords in the unit interval, where each codeword occupies a non-overlapping fraction of length 2^{-l_i} of the unit interval (note that when a codeword of length k is used, a 2^{-k} fraction of the associated binary tree is "taken", because of the prefix-free condition).

Problem 3.6

Suppose that Alice wants to build a prefix-free code such that $l_1 = l_4 = 2$, $l_2 = l_3 = 3$ and $l_5 = 1$. Can you show to Alice that this is not possible? (Try both analogies: the binary tree and the unit interval covering)

Discussion topic 3.3

Given a set of lengths, l_i, in increasing order, discuss strategies for covering the unit interval as efficiently as possible. Can the covering be done as efficiently as previously even if the lengths are given in an arbitrary order?

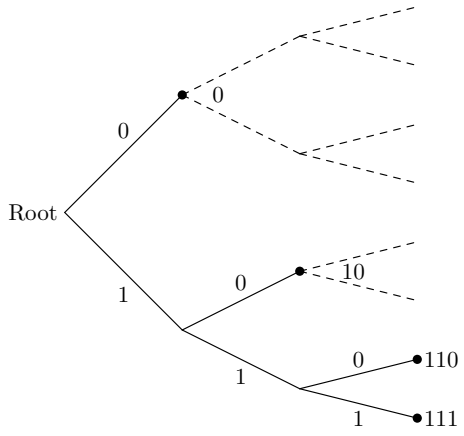


Figure 3.4: Tree associated to a prefix-free code.

Theorem 3.1 (Kraft inequality)
For any instantaneous (i.e., prefix-free) binary code, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy the inequality

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1.$$

Proof. Referring to the code tree of Fig. 3.4, consider all nodes at level $l_{\max}$ (the length of the longest codeword). A codeword at level l_i has $2^{l_{\max}-l_i}$ descendants at level $l_{\max}$. Because of the prefix-free condition, each of these descendant sets must be disjoint. Also, the total number of nodes in these sets must be less than or equal to $2^{l_{\max}}$. Therefore, summing over all the codewords, we have $\sum 2^{l_{\max}-l_i} \leq 2^{l_{\max}}$ or $\sum 2^{-l_i} \leq 1$. □

Theorem 3.2 (Counterpart of the Kraft inequality)
If a set of lengths $l_1, l_2, \ldots, l_m$ satisfy the Kraft inequality, then there exists an instantaneous code with these word lengths.

Proof. Construct a tree of depth $l_{\max}$. Label the first node (lexicographically) of depth l_1 as codeword 1 and remove its descendants from the tree. Then label the first remaining node of depth l_2 as codeword 2, and so on. Proceeding this way, we construct a prefix code with the specified $l_1, l_2, \ldots, l_m$. □

3.2 Definition of information according to Shannon

The definition of **information** according to Shannon is associated with the probability of certain events (the symbols). Consider E an event, i.e., a set of outcomes of some random experiment. If $P(E)$ is the probability of event E to occur, then the information (also called by Shannon "self-information") associated with the occurrence of E is given by

$$i(E) = \log_b \frac{1}{P(E)} = -\log_b P(E).$$

Recall that $\log(1) = 0$ and that $-\log(x)$ increases as x decreases from one to zero. Hence, an event with probability one does not carry any information. On the other hand, as the probability of the event approaches zero, the information that can be associated with that event approaches infinity.

Moreover, the information associated with the occurrence of two **independent** events, E_1 and

E_2,[3] is $i(E_1, E_2) = i(E_1) + i(E_2)$, because

$$i(E_1, E_2) = -\log_b P(E_1, E_2) = -\log_b P(E_1)P(E_2) =$$
$$= -\log_b P(E_1) - \log_b P(E_2) = i(E_1) + i(E_2).$$

As mentioned before, the unit of information depends on the base of the logarithm, b: if $b = 2$, we measure information in bits; if $b = e$, in nats; if $b = 10$, in hartleys. From now on, when the base of the logarithm is not specified, we assume $b = 2$.

For a certain partition of the sample space, S, in E_j sets, i.e., if

$$\bigcup_j E_j = S \quad \text{and} \quad E_j \cap E_k = \emptyset, \forall j \neq k,$$

the associated **average self-information** is given by

$$H = \sum_j P(E_j) i(E_j) = -\sum_j P(E_j) \log P(E_j),$$

more usually called **entropy**. Note that the entropy is completely defined by a probability distribution—hence the name "**probabilistic**" in this definition of information.

Example 3.3
Consider tossing one fair coin. In this case, we have:

- *Two symbols (i.e.,* $\Sigma = \{heads, tails\}$*),* $P_i = 0.5$
- $H = -(0.5 \log 0.5 + 0.5 \log 0.5) = 1$
  $\Longrightarrow$ *One bit for each trial*
- $R = \log |\Sigma| - H = \log 2 - 1 = 0$
  This is called ***redundancy*** *and measures the* ***difference between the combinatorial and probabilistic information measures****.*

Example 3.4
The case of two fair and independent coins:

- *Four symbols,* $P_i = 0.25$
- $H = -\sum_{i=1}^{4} P_i \log P_i = 2$ *bits*

[3] Recall that if E_1 and E_2 are independent, then $P(E_1, E_2) = P(E_1)P(E_2)$.

- $R = \log|\Sigma| - H = \log 4 - 2 = 0$

Example 3.5
Two independent, but biased, coins:

- *Four symbols,* $P_1 = 0.5625$*,* $P_2 = P_3 = 0.1875$ *and* $P_4 = 0.0625$
- $H \approx 1.62$ *bits*
- $R = \log|\Sigma| - H = \log 4 - 1.62 = 0.38$
- *In this case, representing each symbol, on average requires only* 1.62 *bits.*
- *This means that, for example,* 1000 *outcomes of the process can be represented by approximately* 1620 *bits, instead of* 2000 *bits, as suggested by the combinatorial definition of information.*
- *As already mentioned, this data reduction can be obtained using* ***variable-length codes****.*

Shannon demonstrated that, if an **information source** produces events with probabilities $P(E_i)$, where $\{E_i\}$ is a partition of S, then, **on average**, it is not possible to use less than H bits to represent each event.

Usually, and for simplicity, we label the events E_i with symbols from an alphabet Σ which is often a subset of the integers, for example, $\Sigma = \{1, 2, \ldots, m\}$. However, for the sake of generality, we will use the more generic $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ alphabet.

Let us assume that an information source produces sequences of symbols $\mathbf{X} = \{X_1, X_2, \ldots\}$,[4] with $X_i \in \Sigma$. The nth-order entropy, H_n, is defined as

$$H_n = -\frac{1}{n} \sum_{x_1=\sigma_1}^{\sigma_m} \cdots \sum_{x_n=\sigma_1}^{\sigma_m} P(X_{k+1} = x_1, \ldots, X_{k+n} = x_n) \log P(X_{k+1} = x_1, \ldots, X_{k+n} = x_n),$$

where $\{X_{k+1}, X_{k+2}, \ldots, X_{k+n}\}$ are sub-sequences of length n generated by the information source. Shannon showed that, for a **stationary source**,

$$H = \lim_{n\to\infty} H_n,$$

where H is the **entropy of the source**. Notice that, if the events are **independent and identically distributed** (i.i.d.), then

$$H = H_1 = H_n = -\sum_{x=\sigma_1}^{\sigma_m} P(X_k = x) \log P(X_k = x), \quad \forall_k, \tag{3.2}$$

i.e., the entropy and the 1st-order entropy of the source are the same.

[4] $\mathbf{X}$ is a discrete-time random process and, therefore, the X_i are random variables.

Problem 3.7
Show that (3.2) holds $\forall_n$, if the events are independent and identically distributed.

Generally, the entropy of an information source is not known. Therefore, we have to compute **estimates** for it.

3.3 Source modeling

Consider the following sequence of symbols, produced by a certain information source:

```
1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10
```

If we take it as a "good" representation of the statistics of the information source, then we can estimate the probabilities of the symbols as

$$P(1) = P(6) = P(7) = P(10) = 1/16$$
$$P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = 2/16.$$

Assuming i.i.d., the entropy of this source is 3.25 bits.

However, if we transform the original sequence,

```
1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10
```

into

```
1 1 1 -1 1 1 1 -1 1 1 1 1 1 -1 1 1
```

we now have only two different symbols (-1 and 1), for which $P(1) = 13/16$ and $P(-1) = 3/16$. Now, the entropy is only 0.7 bits...

Discussion topic 3.4
We have seen that the entropy is a measure of the average of information produced by a source. According to the example above, apparently it is possible to change the amount of information produced by a source, using a reversible transformation (show that the transformation used above is, in fact, reversible).

Do you think this is indeed possible? [5]

Let us denote by $x_1^n = x_1 x_2 \dots x_n,\ x_i \in \Sigma$, the sequence of outputs (symbols from the source alphabet Σ) that an information source has generated until instant n. Often, we refer to x_1^n (or

[5] In fact, it is not! So, what is wrong in this reasoning?

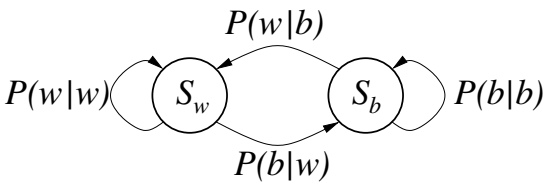


Figure 3.5: Example of a transition diagram of a Markov model for a binary image.

simply x, if size can be omitted) as a string over the alphabet Σ. The transformation that was performed in the example above corresponds to the observation that the relation

$$x_n = x_{n-1} + r_n$$

provides a "satisfactory" **model** for this information source. In that case, only the values of r_n need to be encoded. This model is called **static**, because it does not change with n. Otherwise, the model is called **adaptive**.

Discussion topic 3.5
How do we know that a certain model is "satisfactory"? For example, why is it resonable to say that the model above is "satisfactory"?

Correctly modeling an information source (and, therefore, **finding the underlying structure of the data**) is one of the most important aspects in data compression.

One of the most used aproaches for representing data dependencies relies on the use of **Markov models**. In lossless data compression, we use a specific type, called discrete time Markov chain or **finite-context model**.

A k-order Markov model verifies

$$P(x_n|x_{n-1}\dots x_{n-k}) = P(x_n|x_{n-1}\dots x_{n-k}\dots),$$

where the string $c = x_{n-1}\dots x_{n-k}$ is called the state or **context** of the process.

A 1st-order Markov model reduces to

$$P(x_n|x_{n-1}) = P(x_n|x_{n-1}x_{n-2}\dots).$$

As an example, consider that we want to model a binary image. We have two states, S_w (white pixel) and S_b (black pixel), and four possible transitions, namely $S_w \to S_w$, $S_w \to S_b$, $S_b \to S_w$, $S_b \to S_b$. The state transition diagram of this 1st-order model is shown in Fig. 3.5.

The entropy of a process with N states S_i is simply the average value of the entropy of each state, i.e.,

$$H = \sum_{i=1}^{N} P(S_i)H(S_i),$$

where $H(S_i)$ denotes the entropy of state S_i and P_i is the probability of occurrence of state S_i. Therefore, in our example,

$$H(S_w) = -P(b|w)\log P(b|w) - P(w|w)\log P(w|w).$$

Example 3.6
Consider a 1st-order model with

$$P(w|b) = 0.3 \quad \text{and} \quad P(b|w) = 0.01.$$

The state probabilities can be calculated according to

$$P(S_w) = \frac{P(w|b)}{P(w|b) + P(b|w)} \approx 0.968$$

and

$$P(S_b) = \frac{P(b|w)}{P(w|b) + P(b|w)} \approx 0.032.$$

Therefore, considering independence in the occurrence of symbols, we have,

$$H = -0.968\log_2 0.968 - 0.032\log_2 0.032 \approx 0.204 \textit{ bps}$$

However, considering the 1st-order Markov model, we obtain an estimate of the entropy of the source that is

$$H(S_b) = -0.3\log_2 0.3 - 0.7\log_2 0.7 \approx 0.881 \textit{ bps}$$
$$H(S_w) = -0.01\log_2 0.01 - 0.99\log_2 0.99 \approx 0.081 \textit{ bps}$$
$$H = 0.968 \times 0.081 + 0.032 \times 0.881 \approx 0.107 \textit{ bps}$$

Markov models are particularly useful in text compression, because the next letter in a word is generally heavily influenced by the preceding letters. In fact, the use of Markov models for written English appeared in the original work of Shannon. In 1951, he estimated the entropy of English to be in between about 0.6 and 1.3 bits per letter.

For simplicity, consider only 26 letters and the space character. The frequency of letters in English is far from uniform: the most common, "E", occurs about 13%, whereas the least common, "Q" and "Z", occur about 0.1% of the time. The frequency of pairs of letters is also nonuniform. For example, the letter "Q" is always followed by a "U". The most frequent pair is "TH" (occurs about 3.7%).

We can use the frequency of the pairs to estimate the probability that a letter follows any other letter. This reasoning can also be used for constructing higher-order models. However, the size of the model grows exponentially. For example, to build a third-order Markov model we need to estimate the values of $p(x_n|x_{n-1}x_{n-2}x_{n-3})$. This requires a table with $27^4 = 531\,441$ entries and enough text to correctly estimate the probabilities.

The following examples have been constructed using empirical distributions collected from samples of text (Shannon, 1948, 1951).

Example 3.7 (Zero-order approximation (Independent and equiprobable symbols))

XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGXYD QPAAMKBZAACIBZLHJQD

Entropy: $\log 27 = 4.76$ *bits per letter.*

Example 3.8 (First-order approximation (Independent, but with the correct $p(x)$))

OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL

Estimated entropy: ≈ 4.03 *bits per letter.*

Example 3.9 (Second-order approximation (1st-order Markov model, i.e., $p(x_n|x_{n-1})$))

ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE

Estimated entropy: ≈ 3.32 *bits per letter.*

Example 3.10 (Third-order approximation (2nd-order Markov model, i.e., $p(x_n|x_{n-1}x_{n-2})$))

IN NO IST LAT WHEY CRATICT FROURE BERS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE

Estimated entropy: ≈ 3.1 *bits per letter (excluding spaces).*

Example 3.11 (First-order word approximation)

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

Example 3.12 (Second-order word approximation)

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

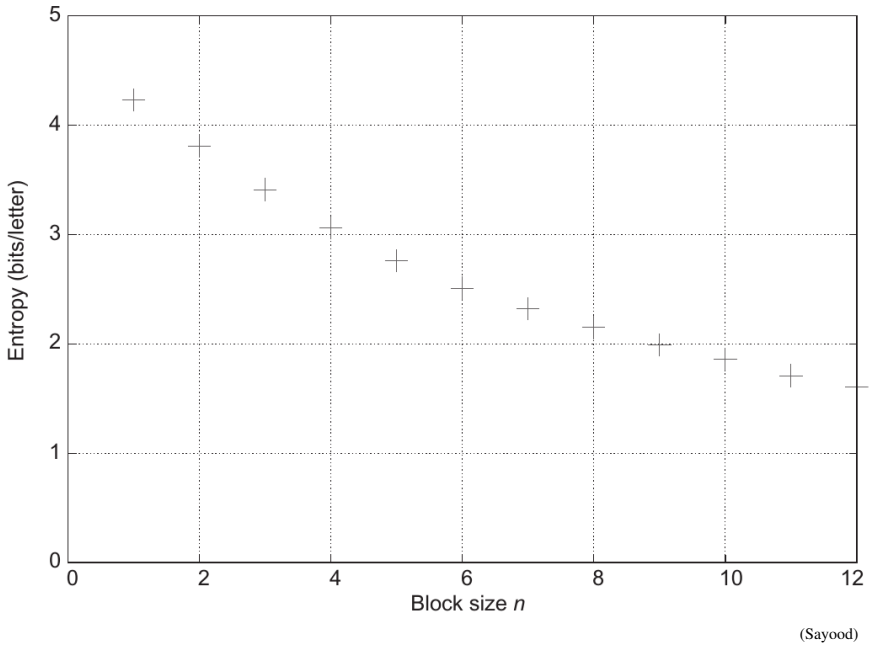


Figure 3.6: Estimated value of the entropy of a text using serveral orders.

Figure 3.6 shows the evolution of the estimate of the entropy of a book when the order of the model increaces.

The number of different contexts (or states of the Markov process) is given by $|\Sigma|^k$, where k is the order of the model, which clearly grows exponentially with the size of the alphabet. For example, in a 2-symbol alphabet, a finite-context model of depth 4 originates only $2^4 = 16$ states or different contexts. However, if the alphabet is of size 256 (for example, if we consider a gray-level image), then the number of contexts jumps to $256^4 = 4\,294\,967\,296$!

Often, the finite-context models are "learned" online, i.e., as the sequence of symbols is processed (note that in Example 3.6 the model is static, i.e., the probabilities are considered fixed during the operation of the model). In the case of online operation, since the probabilistic models are continuously adapting, after processing the first n symbols of x, the average number of bits associated with an order-k finite-context model is given by

$$H_n = -\frac{1}{n}\sum_{i=1}^{n} \log P(x_i|x_{i-k}^{i-1}). \tag{3.3}$$

Problem 3.8
Verify (3.3) and discuss the practical implications of x_{n-k}^{n-1} when $n < k$ (note that the convention is that $x_n^n = x_n$) and propose solutions to handle it.

In the case of online learning of the finite-context model, a table (or some other more sophisticated data structure, such as an hash-table) is used to collect counts that represent the number

of times that each symbol occurs in each context. For example, suppose that, in a certain moment, a binary ($|\Sigma| = 2$) source is modeled by an order-3 finite-context model represented by the table

x_{i-3}	x_{i-2}	x_{i-1}	n_0	n_1
0	0	0	10	25
0	0	1	4	12
0	1	0	15	2
0	1	1	3	4
1	0	0	34	78
1	0	1	21	5
1	1	0	17	9
1	1	1	0	22

where n_i indicates how many times symbol "i" occurred following that context. Therefore, in this case, we may estimate the probability that a symbol "0" follows the string "100" as being $34/(34 + 78) \approx 0.30$.

This way of estimating probabilities, based only on the relative frequencies of previously occurred events, suffers from the problem of assigning probability zero to events that were not (yet) seen. That would be the case, in this example, if we used the same approach to estimate the probability of having a "0" following a string of three or more "1s".

Problem 3.9
Discuss the potential implications of the "zero-probability" problem and give suggestions to avoid it.

In 1774, Laplace (1749–1827) proposed a rule for estimating the probability that a given event succeeds, after occurring n_1 successes and n_2 failures in $n = n_1 + n_2$ trials, as

$$\frac{n_1 + 1}{n + 2}.$$

This formula is in fact correct if the trials are independent and if an uniform prior over the estimated probability is considered.

For the more general case of a beta prior, $B(\alpha_1, \alpha_2)$, the estimator adopts the form (G. F. Hardy, 1889)

$$\frac{n_1 + \alpha_1}{n + \alpha_1 + \alpha_2}.$$

The multinomial generalization of the formula leads to

$$\frac{n_i + \alpha}{n + \alpha|\Sigma|},$$

where, for simplicity, we consider $\alpha = \alpha_1 = \cdots = \alpha_{|\Sigma|}$, and the α_i are the parameters of the Dirichlet family of distributions.

A worth noting aspect of this estimator is that, defining

$$\mu = \frac{n}{n + \alpha|\Sigma|},$$

it can be written as

$$\mu\frac{n_i}{n} + (1-\mu)\frac{1}{|\Sigma|},$$

showing how it evolves from a uniform estimator to a frequency estimator, as n increases.

3.4 Some more on Shannon entropy

3.4.1 Entropy, joint entropy and conditional entropy

As already shown, given a probability distribution, we may define a quantity called **entropy**, that has many properties that agree with our intuitive notion of what a measure of information should be. We have seen that the entropy expresses the **amount of uncertainty** about a certain random variable. It measures the **amount of information** required **on average** to describe the random variable.

This notion can be extended to define **mutual information**, which is a measure of the amount of information that one random variable contains about another. In fact, the mutual information is a special case of a more general quantity called **relative entropy**, which gives a measure of the "distance" between two probability distributions.

Consider X a discrete random variable that takes values in a finite alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ and has a probability mass function $p(x) = P(X = x)$, $x \in \Sigma$.

The entropy of X can be interpreted as the **expected value** of the random variable $\log \dfrac{1}{p(X)}$, where X is drawn according to $p(x)$. Therefore,

$$H(X) = E\left[\log \frac{1}{p(X)}\right] = E[-\log p(X)],$$

where $E[g(X)]$ denotes the expected value of $g(X)$, defined as

$$E[g(X)] = \sum_{x\in\Sigma} g(x)p(x).$$

Example 3.13
Let X be a binary random variable with alphabet $\Sigma = \{0, 1\}$ and probability mass function $P(X = 1) = p$ and $P(X = 0) = 1 - p$. Then, the entropy of X is $H(X) = -p\log p - (1 - p)\log(1 - p)$ (see Fig. 3.7).

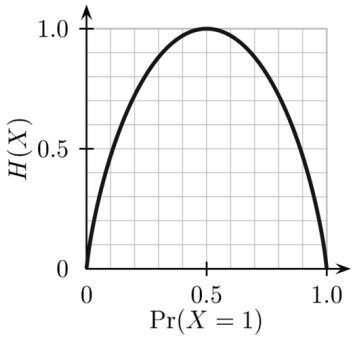


Figure 3.7: Entropy curve for a binary alphabet.

Theorem 3.3
$0 \leq H(X) \leq \log|\Sigma|$, *where* $|\Sigma|$ *denotes the cardinality of* Σ. *Also,* $H(X) = 0$ *iff* $p(x) = 1$ *for some* $x \in \Sigma$, *and* $H(X) = \log|\Sigma|$ *iff* $p(x) = 1/|\Sigma|$.

Proof of $H(X) \geq 0$. Since each $p(x) \leq 1$, then each term $p(x)\log p(x) \leq 0$, implying $H(X) \geq 0$. Also, $H(X) = 0$ requires that $p(x)\log p(x) = 0, \forall x \in \Sigma$, implying $p(x) = 1$ or $p(x) = 0$.

Therefore, $H(X) = 0$ iff $p(x) = 1$ for some $x \in \Sigma$ (and, obviously, $p(x) = 0$ for the other x). □

Proof of $H(X) \leq \log|\Sigma|$. Consider the inequality $\ln x \leq x - 1$ and the subset $\Sigma' \subset \Sigma$ such that $p(x) > 0, \forall x \in \Sigma'$. Then

$$H(X) - \log|\Sigma'| = \sum_{x\in\Sigma'} p(x)\log\frac{1}{p(x)} - \sum_{x\in\Sigma'} p(x)\log|\Sigma'| =$$

$$= \frac{1}{\ln 2}\sum_{x\in\Sigma'} p(x)\ln\frac{1}{|\Sigma'|p(x)} \leq \frac{1}{\ln 2}\sum_{x\in\Sigma'} p(x)\left[\frac{1}{|\Sigma'|p(x)} - 1\right] =$$

$$= \frac{1}{\ln 2}\left[\sum_{x\in\Sigma'}\frac{1}{|\Sigma'|} - \sum_{x\in\Sigma'} p(x)\right] = 0.$$

The rest of the proof is left as exercise. □

We can extend the definition of entropy to a pair of random variables (X, Y) with joint distribution $p(x, y) = P(X = x, Y = y)$. For simplicity, we will assume that both random variables are drawn from the same alphabet Σ.

Definition 3.1 (Joint entropy)
The joint entropy $H(X,Y)$ of a pair of discrete random variables (X,Y), with a joint distribution $p(x,y)$, is defined as

$$H(X,Y) = -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x,y) = E[-\log p(X,Y)].$$

Note that if $p(x,y) = p(x)p(y)$ (i.e., if X and Y are independent random variables), then $H(X,Y) = H(X) + H(Y)$. On the other hand, if X and Y are dependent, then $H(X,Y) < H(X) + H(Y)$.

Definition 3.2 (Conditional entropy)
For a pair of discrete random variables (X,Y), having a joint distribution $p(x,y)$, we define conditional entropy $H(Y|X)$ as

$$H(Y|X) = \sum_{x\in\Sigma} p(x)H(Y|X=x) = -\sum_{x\in\Sigma} p(x)\sum_{y\in\Sigma} p(y|x)\log p(y|x) =$$

$$= -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(y|x) = E[-\log p(Y|X)].$$

Theorem 3.4 (Chain rule for the entropy of a pair of random variables)

$$H(X,Y) = H(X) + H(Y|X).$$

Proof.

$$H(X,Y) = -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x,y) = -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x)p(y|x) =$$

$$= -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x) - \sum_{(x,y)\in\Sigma^2} p(x,y)\log p(y|x) =$$

$$= -\sum_{x\in\Sigma} p(x)\log p(x) + H(Y|X) = H(X) + H(Y|X).$$

□

Example 3.14

Consider the following joint distribution of (X, Y)*:*

$Y \backslash X$	1	2	3	4
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
4	$\frac{1}{4}$	0	0	0

The marginal distributions of X *and* Y *are, respectively,* $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ *and* $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$*. Hence,* $H(X) = 1.75$ *bits and* $H(Y) = 2$ *bits. Also,*

$$H(X|Y) = \sum_{i=1}^{4} P(Y=i)H(X|Y=i) = \frac{1}{4}H\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right) +$$

$$\frac{1}{4}H\left(\frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{8}\right) + \frac{1}{4}H\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) + \frac{1}{4}H(1,0,0,0) = 1.375 \text{ bits.}$$

Similarly, $H(Y|X) = 1.625$ *bits and* $H(X,Y) = 3.375$ *bits.*

Note that, generally, $H(Y|X) \neq H(X|Y)$, but that $H(X) - H(X|Y) = H(Y) - H(Y|X)$.

3.4.2 Relative entropy

The relative entropy is a measure of the "distance" between two distributions. It expresses the inefficiency of assuming that the distribution is $q(x)$ when the true distribution is $p(x)$.

Definition 3.3 (Relative entropy)
The relative entropy or Kullback-Leibler distance between two probability mass functions, $p(x)$ *and* $q(x)$*, is defined as*

$$D(p\|q) = \sum_{x \in \Sigma} p(x) \log \frac{p(x)}{q(x)} = E_p\left[\log \frac{p(X)}{q(X)}\right].$$

Note that we consider the conventions: $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$.

Example 3.15
Let $\Sigma = \{0, 1\}$ *and consider two distributions,* $p(x)$ *and* $q(x)$*, on the elements of* Σ*. Let* $p(0) = 1 - r$*,* $p(1) = r$*,* $q(0) = 1 - s$ *and* $q(1) = s$*. Then*

$$D(p\|q) = (1-r) \log \frac{1-r}{1-s} + r \log \frac{r}{s}$$

and

$$D(q\|p) = (1-s)\log\frac{1-s}{1-r} + s\log\frac{s}{r}$$

If $r = 1/2$ *and* $s = 1/4$*, then* $D(p\|q) = 0.208$ *bits,* $D(q\|p) = 0.189$ *bits.*

Note that, in general, $D(p\|q) \neq D(q\|p)$, and, if $p(x) = q(x)$, then $D(p\|q) = D(q\|p) = 0$.

3.4.3 Mutual information

The mutual information expresses the amount of information that one random variable contains about another rv.

Definition 3.4 (Mutual information)
Consider two random variables X *and* Y*, with joint probability mass function* $p(x,y)$ *and marginal functions* $p(x)$ *and* $p(y)$*.*

The mutual information, $I(X;Y)$*, can be defined as the relative entropy between the joint distribution and the product* $p(x)p(y)$*, i.e.,*

$$I(X;Y) = D(p(x,y)\|p(x)p(y)) =$$

$$= \sum_{(x,y)\in\Sigma^2} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} = E_{p(x,y)}\left[\log\frac{p(X,Y)}{p(X)p(Y)}\right].$$

Also, the mutual information indicates the reduction in the uncertainty of one random variable due to the knowledge of the other. In fact, we can write

$$I(X;Y) = \sum_{(x,y)\in\Sigma^2} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} = \sum_{(x,y)\in\Sigma^2} p(x,y)\log\frac{p(x|y)}{p(x)} =$$

$$= -\sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x) + \sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x|y) =$$

$$= -\sum_{x\in\Sigma} p(x)\log p(x) + \sum_{(x,y)\in\Sigma^2} p(x,y)\log p(x|y) =$$

$$= H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y).$$

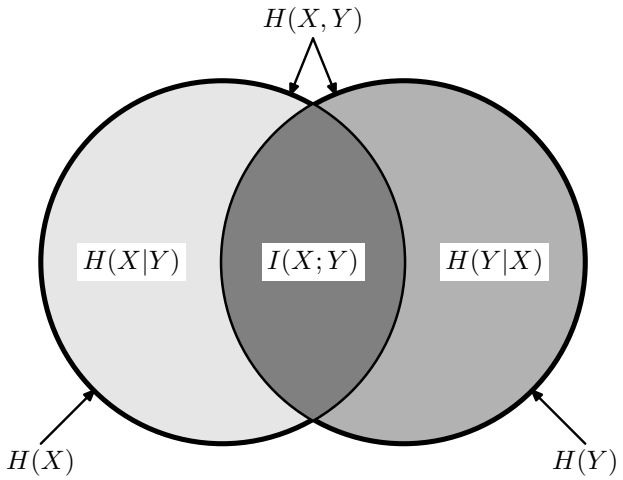


Figure 3.8: Some relations between entropy and mutual information.

Example 3.16
Considering again the joint distribution of (X, Y)*,*

$Y \backslash X$	1	2	3	4
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
4	$\frac{1}{4}$	0	0	0

where $H(X) = 1.75$ *bits,* $H(Y) = 2$ *bits,* $H(X|Y) = 1.375$ *bits and* $H(Y|X) = 1.625$*, we obtain a mutual information between* X *and* Y *of*

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = 0.375 \quad \text{bits.}$$

Theorem 3.5 (Relations between the mutual information and the entropy)
The following relations between the mutual information of a pair of random variables, X *and* Y*, and the corresponding entropies hold (see also Fig. 3.8):*

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) \\
I(X;Y) &= H(Y) - H(Y|X) \\
I(X;Y) &= H(X) + H(Y) - H(X,Y) \\
I(X;Y) &= I(Y;X) \\
I(X;X) &= H(X) - H(X|X) = H(X).
\end{aligned}$$

The mutual information of a random variable with itself is the entropy of the random variable (verify this). This is why the entropy is sometimes referred to as **self-information**.

Theorem 3.6 (Chain rule for the entropy)
Let $X_1, X_2, \ldots, X_n$ be a set of random variables drawn according to $p(x_1, x_2, \ldots, x_n)$. Then

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1).$$

Proof.

$$H(X_1, X_2) = H(X_1) + H(X_2|X_1),$$

$$H(X_1, X_2, X_3) = H(X_1) + H(X_2, X_3|X_1) = H(X_1) + H(X_2|X_1) + H(X_3|X_2, X_1),$$

$$\vdots$$

$$H(X_1, X_2, \ldots, X_n) = H(X_1) + H(X_2|X_1) + \cdots + H(X_n|X_{n-1}, \ldots, X_1).$$

□

Definition 3.5 (Conditional mutual information)
The conditional mutual information of the random variables X and Y given Z is defined as

$$I(X;Y|Z) = H(X|Z) - H(X|Y,Z) = E_{p(x,y,z)}\left[\log \frac{p(X,Y|Z)}{p(X|Z)p(Y|Z)}\right].$$

(Remember that $I(X;Y) = H(X) - H(X|Y)$)

Theorem 3.7 (Chain rule for the information)

$$I(X_1, X_2, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y|X_{i-1}, X_{i-2}, \ldots, X_1).$$

Proof.

$$\begin{aligned} I(X_1, X_2, \ldots, X_n; Y) &= H(X_1, X_2, \ldots, X_n) - H(X_1, X_2, \ldots, X_n|Y) = \\ &= \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1) - \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1, Y) = \\ &= \sum_{i=1}^{n} I(X_i; Y|X_{i-1}, X_{i-2}, \ldots, X_1). \end{aligned}$$

□

Theorem 3.8 (Jensen's inequality)
If $f(x)$ is a convex function and X is a random variable,

$$E[f(X)] \geq f(E[X]).$$

Moreover, if $f(x)$ is strictly convex, the equality implies that $X = E[X]$ with probability 1, i.e., X is constant.

Theorem 3.9 (Information inequality)
Let $p(x)$ and $q(x)$ be two probability mass functions, with $x \in \Sigma$. Then, with equality iff $p(x) = q(x), \forall x$,

$$D(p\|q) \geq 0.$$

Proof of the non-negativity. Let $\Sigma' = \{x|p(x) > 0\}$ be the support set of $p(x)$. Then

$$-D(p\|q) = \sum_{x\in\Sigma'} p(x) \log \frac{q(x)}{p(x)} \leq \quad \text{(by Jensen inequality)}$$

$$\leq \log \sum_{x\in\Sigma'} p(x)\frac{q(x)}{p(x)} = \log \sum_{x\in\Sigma'} q(x) \leq \log \sum_{x\in\Sigma} q(x) = \log 1 = 0,$$

i.e., $D(p\|q) > 0$. □

Proof that $D(p\|q) = 0$ iff $p = q$. Since $\log x$ is a strictly concave function of x, we have

$$\sum_{x\in\Sigma'} p(x) \log \frac{q(x)}{p(x)} = \log \sum_{x\in\Sigma'} p(x)\frac{q(x)}{p(x)}$$

iff $q(x)/p(x)$ is constant everywhere (i.e., if $q(x) = cp(x), \forall x$). Therefore,

$$\sum_{x\in\Sigma'} q(x) = c \sum_{x\in\Sigma'} p(x) = c.$$

Moreover, it is required that $\sum_{x\in\Sigma'} q(x) = 1$ for having

$$\log \sum_{x\in\Sigma'} q(x) = \log \sum_{x\in\Sigma} q(x),$$

implying $c = 1$. Hence, we have $D(p\|q) = 0$ iff $p(x) = q(x), \forall x$. □

Corollary 3.9.1 (Nonnegativity of mutual information)
For any two random variables, X and Y,

$$I(X;Y) \geq 0,$$

with equality iff X and Y are independent.

Proof.

$$I(X;Y) = D(p(x,y)\|p(x)p(y)) \geq 0,$$

with equality iff $p(x,y) = p(x)p(y)$, i.e., when X and Y are independent. □

Theorem 3.10 (Upper bound on the entropy)

$$H(X) \leq \log |\mathcal{X}|,$$

with equality iff X has uniform distribution over $\mathcal{X}$.

Proof. Let $u(x) = 1/|\Sigma|$ be the uniform probability mass function over Σ, and let $p(x)$ be the probability mass function for X. Then,

$$D(p\|u) = \sum_{x\in\Sigma} p(x) \log \frac{p(x)}{u(x)} = \log |\Sigma| - H(X) \geq 0.$$

Therefore, $H(X) \leq \log |\Sigma|$, with equality iff $p(x) = u(x)$. □

Theorem 3.11 (Conditioning reduces entropy)

$$H(X|Y) \leq H(X)$$

with equality iff X and Y are independent.

Proof.

$$0 \leq I(X;Y) = H(X) - H(X|Y).$$

□

This theorem states that knowing another random variables, Y, can only reduce the uncertainty in X. However, this is only true **on average**. Specifically, $H(X|Y = y)$ may be greater than or less than or equal to $H(X)$, but, on average, $H(X|Y) = \sum_y p(y)H(X|Y = y) \leq H(X)$.

Example 3.17
Let (X, Y) have the following joint distribution,

$Y\backslash X$	1	2
1	0	$\frac{3}{4}$
2	$\frac{1}{8}$	$\frac{1}{8}$

Then, $H(X) = H(\frac{1}{8}, \frac{7}{8}) \approx 0.544$ *bits,* $H(X|Y=1) = 0$ *and* $H(X|Y=2) = 1$ *bit. However,*

$$H(X|Y) = \frac{3}{4}H(X|Y=1) + \frac{1}{4}H(X|Y=2) = 0.25 \quad \text{bits.}$$

Thus, the uncertainty in X *increases if* $Y=2$ *is observed and decreases if* $Y=1$*. Nevertheless, on average, the uncertainty decreases.*

Theorem 3.12 (Independence bound on the entropy)
Let $X_1, X_2, \ldots, X_n$ *be drawn according to* $p(x_1, x_2, \ldots, x_n)$*. Then,*

$$H(X_1, X_2, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i),$$

with equality iff the X_i *random variables are independent.*

Proof. By the chain rule for entropies and using the fact that conditioning reduces the entropy,

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1) \leq \sum_{i=1}^{n} H(X_i),$$

with equality iff the X_i are independent. □

---

4 An algorithmic measure of information

4.1 Motivation

As shown previously, the formulation of a probabilistic measure of information explores statistical regularities of the data, by means of probabilistic modeling of the information sources. However, there are regularities that cannot be captured by probabilistic models alone. For example, several approaches involving the statistical analysis of the initial digits of π have failed to reveal significant deviations from what is usually understood as statistical **randomness**. Nevertheless, the initial sequence of digits of π can be generated by simple programs, showing that the number π is far from being random! In fact, intuitively, we may say that a string is random if it cannot be **compressed**, i.e., if it is not possible to find a **shorter description** for it. This is precisely the core idea behind the algorithmic measure of information.

Discussion topic 4.1
In data compression, what is the role of the decompressor and of the string representing the compressed data? Does it make sense to consider the decompressor as an "interpreter" for a certain language, formed by all possible compressed strings?

Example 4.1
Consider the following three binary strings:

1. 0101010101010101010101010101010101010101010101010101010101010101

2. 0110101000001001111001100110011111111001110111100110010010000 1000

3. 1101111001110101111101101111101110101101111000101110010100111011

What should be the shortest descriptions for them? The first one seems to be easy to describe. Apparently, the second one looks random, but actually it is just the initial binary representation of $\sqrt{2}-1$. The third one seems random and probably is (however, in general, it is impossible to prove it!).

Now, consider the representation of a sequence generated by flipping a fair coin n times, such as

$$00100101110101010\ldots010$$

There are 2^n such sequences and they are equally probable. It is highly likely that such sequence cannot be compressed at all. Therefore, there might not be a better (shorter) program for generating this sequence as "print $00100101110101010\ldots010$". Thus, the descriptive complexity of a random binary sequence is as long as the sequence itself.

4.2 Turing machines

One question that may arise when thinking about computers is: **Can a computer solve any type of problem?** And, if the answer to this question is "no": Which are the problems that cannot be solved by computers?

In fact, according to Nigel J. Cutland (1944–), in his book "*Computability: An introduction to recursive function theory*", Cambridge University Press (Cutland, 1980),

> *"We could describe computability theory, from the viewpoint of computer science, as beginning with the question What can computers do in principle (without restrictions of space, time or money)?—and, by implication—What are their inherent theoretical limitations?"*

Hence, one of the objectives of the field of computability theory is precisely to study which problems are solvable by computers and which are not. However, to achieve this goal, we need first to have a precise definition of what do we mean by a **digital computer**. In this section, we give a brief overview of some important computation models.

First, let us introduce some notation. We define an **alphabet** as a nonempty finite set. We refer to the members of the alphabet as **symbols**. Usually, and without loss of generality, we consider the alphabet $\Sigma = \{0, 1\}$. A **string** over an alphabet is a finite sequence of symbols from that alphabet. For example, if $\Sigma = \{0, 1\}$, then 011101 is a string over the alphabet Σ. If s is a string, then we represent by $|s|$ the length (i.e., number of symbols) of s. We denote by Σ^* the set of all finite-length strings, i.e.,

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\},$$

where ϵ denotes the empty string, i.e., $|\epsilon| = 0$.

For convenience, we assume an ordering of the strings, such that they appear ordered by size and, for the same size, ordered lexicographically (this is the ordering used to display the set above). Note that one such ordering allows to put all finite-length strings into a one-to-one correspondence with the integers and, hence, to treat strings as integers.

A **language**, A, is a subset of Σ^*, i.e., $A \subseteq \Sigma^*$. A language is **prefix-free** if no string of the language is a proper prefix of another string of the language.

A **Turing machine** is composed of a control based on a finite, non-empty, set of states and access to an unlimited and unrestricted amount of memory (Fig. 4.1).

A Turing machine is a mathematical model (or abstraction) of a hypothetical computing device where:

1. At each step, the machine is in one of a predefined finite (non-empty) set of states.
2. Symbols from a predefined finite (non-empty) set can be both read from and written on a tape (see Fig. 4.2, for an example).

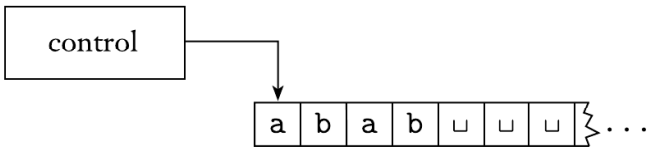


Figure 4.1: Schematic representation of a Turing machine.

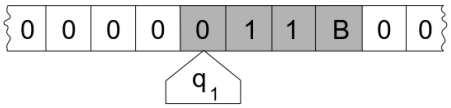


Figure 4.2: Example of a tape and head of a Turing machine. The q_1 written inside the head indicates the current internal state of the machine.

3. The read-write head can move both to the left and to the right.

4. The tape is unlimited.

5. At start, the tape is blank, except for some finite number of squares.

6. The machine stops when a special state (the halting state) is encountered or when it fails to find a continuation state.

7. The special states for rejecting and accepting strings of a language, that we have seen when the automata were addressed, in the case of Turing machines take effect immediately after halting, even without having exhausting the input. There might be a specific halting state for accepting and another one for rejecting, or just a single halting state and the interpretation of acceptance or rejection left written on the tape.

According to Marvin Minsky (1927-2016), in his book "*Computation: Finite and Infinite Machines*", Prentice-Hall, Inc. (Minsky, 1967):

> *"A Turing machine is a finite-state machine associated with an external storage or memory medium. This medium has the form of a sequence of squares, marked off on a linear tape. The machine is coupled to the tape through a head, which is situated, at each moment, on some square of the tape (Fig. 4.2). The head has three functions, all of which are exercised in each operation cycle of the finite-state machine. These functions are: reading the square of the tape being "scanned," writing on the scanned square, and moving the machine to an adjacent square (which becomes the scanned square in the next operation cycle)."*

A Turing machine equipped with accepting and rejecting states, started with a certain string written on the tape, may have one of three possible behaviours: it stops at an accepting state; it stops at a rejecting state; it loops forever.

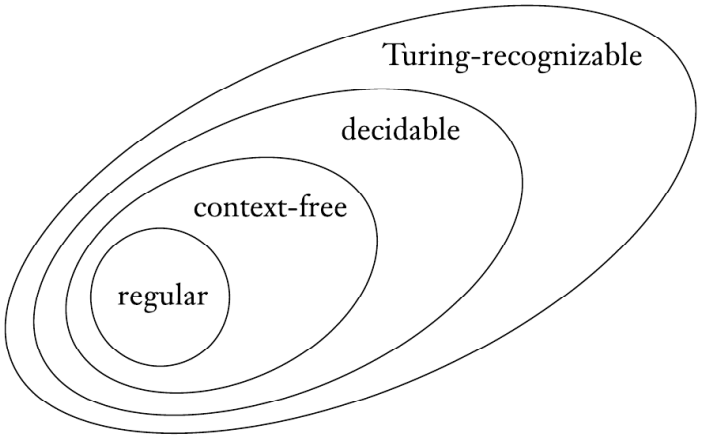


Figure 4.3: Hierarchy of the classes of languages (Sipser, 2012).

Definition 4.1
*A language is **Turing-recognizable / recursively enumerable** if some Turing machine recognizes it.*

This means that a Turing machine recognizes a language if it accepts every string in the language and rejects or does not halt on the strings not in the language.

Definition 4.2
*A language is **Turing-decidable / decidable / recursive**, if some Turing machine decides it.*

This means that a Turing machine decides a language if it accepts every string in the language and rejects every string not in the language. Figure 4.3 shows the hierarchical relation among languages.

There are many variants of Turing machines, for example, using more than one tape/head, using one-way or two-way unlimited tapes, using alphabets of different sizes and with different symbols, and even using nondeterminism. However, all of them share the same computational power.

In fact, there are a number of remarkable facts about Turing machines, some of which are the following:

- A Turing machine can be restricted to a two-symbol alphabet, without lossing any of its computational power. For example, suppose that a certain Turing machine T uses k different symbols and that $n = \lceil \log k \rceil$. Then, we may construct another Turing machine,

T', replacing each symbol of T by its n-bit binary representation and regarding the tape of T' as a sequence of n-bit blocks. Of course, to attain equivalence between T and T' it is also necessary to change appropriately the finite-state machine of T' (which will necessarily have more states than machine T).

- It can also be shown that Turing machines which have tapes that are unlimited in both directions have the same computing power as machines having an unlimited tape in only one direction.
- It is neither an advantage (nor a disadvantage) to have multiple tapes in a Turing machine—it can always be converted into an equivalent single tape machine.
- Probably more surprising is the fact that any Turing machine is equivalent to a Turing machine with only two internal states. Of course, this can only be achieved by enlarging the alphabet of the tape symbols.

The idea behind a Turing machine was first proposed by Alan Turing (1912-1954), in his famous paper:

> Alan Turing, "*On Computable Numbers, with an Application to the Entsgheidungsproblem*", Proceedings of the London Mathematical Society (Turing, 1936).

Almost simultaneously, Alonzo Church (1903-1995) proposed a different, but equivalent, notion of computability, using what became known as λ-calculus:

> Alonzo Church, "*An Unsolvable Problem of Elementary Number Theory*", American Journal of Mathematics (Church, 1936).

In their attempt to solve some problems related to the foundations of mathematics, both had the need to rigorously defining the concept of "computation". Note that, at that time, a "computer" was always seen as a human computer, i.e., someone that performs pencil-and-paper calculations.

Basically, we say that something is **computable** if it can be produced, in finite time, by a process equivalent to our usual intuitive notion of **algorithm** or **effective procedure**. This definition was proposed by Alan Turing in his 1936 paper, where he also refers as equivalent the idea of "effective calculability" of Alonzo Church.

Again, from Minsky (1967):

> *"The idea of an algorithm or effective procedure arises whenever we are presented with a set of instructions about how to behave. This happens when, in the course of working on a problem, we discover that a certain procedure, if properly carried out, will end up giving us the answer. Once we make such a discovery, the task of finding the solution is reduced from a matter of intellectual discovery to a mere matter of effort; of carrying out the discovered procedure—obeying the specified instructions."*

This notion of algorithm motivated a conjecture, known as the **Church-Turing thesis**, stating that something is computable by a human being following an algorithm (of course, ignoring resources, such as time, number of sheets of paper and pencils), if and only if it is computable by a Turing machine. There is no proof for this claim. However, 80 years have passed since the papers of Church and Turing and no counterexample has yet been found!

Example 4.2
Consider the following example of a Turing machine, taken from the original paper of Turing (Turing, 1936). The set of instructions for the machine is:

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
b	*None*	*P0, R*	*c*
c	*None*	*R*	*e*
e	*None*	*P1, R*	*f*
f	*None*	*R*	*b*

If started on an empty tape, the machine writes alternating 0's and 1's on the tape, leaving a blank square between each pair of digits.

Problem 4.1
What happens if, in the previous example, the tape is not totally blank?

One of the most important results included in Turing's paper (Turing, 1936) shows that it is possible to build **universal Turing machines**. A universal Turing machine is a machine that can simulate any other Turing machine. A direct consequence of this discovery is the notion of stored-program computer. All but the most trivial computers are universal, in the sense that they can mimic the actions of other computers. Nevertheless, it is possible to construct a universal standard Turing machine using 4 states and 6 symbols. Moreover, it was recently suggested (although still debated) that a universal Turing machine without a halting state is possible using 2 states and 3 symbols!

One of the most famous problems in computability theory, that cannot be systematically solved by a computer, is to decide if another program, `prog`, will halt for a certain input `x`. Next we show that such program cannot exist.

Let us consider that such program, `will_halt(prog,x)`, exists. Then, consider a program `prog`, such that

```
prog(x)
    if(will_halt(x,x) == true)
        while(true)
```

```
    else
        exit
```

Now, consider what happens if `prog` is called, having as argument a description of itself, i.e., what happens when executing `prog(prog)`. Notice that, inside `prog`, there is a call to `will_halt(prog,prog)`. If this call returns `true`, i.e., if our hypothetical program that decides if a program will halt for a certain input says that it will halt, then `prog` enters an infinite loop and does not halt. On the contrary, if `will_halt(prog,prog)` returns `false`, i.e., if it indicates that `prog` will not stop when called with itself as argument, then it stops. This contradiction shows that it is not possible to build a program that, for all possible cases, verifies if some other program halts or not.

4.3 Kolmogorov complexity (or algorithmic entropy)

Suppose that we want to represent a certain object by a finite string of bits. Generally, an object may have several possible representations (or descriptions), but each representation should describe only one object (we want to unambiguously reconstruct the object from its description). Then, the length of the shortest of all descriptions available for the object can be used as a measure of its **complexity** (and, in fact, of its randomness).

However, the meaning of "**description**" must be defined precisely or otherwise we may run into similar problems as the one known as the **Berry paradox**, where a certain number is defined as "*the smallest positive integer not definable in fewer than twelve words*". If this number exists, then we have just described it in eleven words, which contradicts its definition! On the other hand, if the number does not exist, then we have to conclude that all positive integers can be described in less than twelve words!

The approach that follows relies on algorithms to measure the quantity of information conveyed by a certain string, x. One way to address this problem could involve building a (compact) Turing machine, T_x, that, when started on a blank tape, would write the string x on the tape and halt. Then, the description of x would be a string describing T_x, $\langle T_x \rangle$, using some fixed encoding for the Turing machines and, therefore, the length of the description would be $|\langle T_x \rangle|$.

However, instead of starting the machine on a blank tape, another possible approach is using an auxiliary string written on it. In this case, the string x is described using a Turing machine M and an input w that is used by M, and the description of x is given by some encoding $\langle M, w \rangle$ (note that, in principle, this encoding needs to include information that permits separating $\langle M \rangle$ from w).

Assume that a certain specification method, f, associates at most an object, x, with a description, p. We can think of f as a decompression function (or program), from a set of descriptions, P, into the set of objects, X, i.e., $f : P \to X$. To be useful, descriptions should be finite. This implies that **it is only possible to describe a countable number of different objects**.

For most functions f, the length of the description of object x according to f depends not only on x, but also on f. However, in order to objectively assess the complexity of the objects, we

need functions that assign a complexity value to x that depends on x alone. Moreover, if p is the description of x with respect to f, then x has to be generated from f and p through "**an effective procedure**" or, in other words, f has to be a computable function.

Definition 4.3 (Algorithmic complexity)
Given a computable function, f, we define the algorithmic complexity of an object,[6] *x, according to f, as*

$$K_f(x) = \min\{|p| : f(p) = x\},$$

where $x, p \in \{0,1\}^*$, i.e., it corresponds to the minimum description length, over all descriptions that produce x through f. If there is no p able to produce x using f, then we say that $K_f(x) = \infty$ for such x. Therefore, although computable, f is not required to be a total function.

Notice that we can think of p as a program and f as a (specialized) computer. If $K_f(x) = \infty$, then it is not possible to produce x with the computer f. Also, notice that, without the computability requirement, f could be any arbitrary partial function from strings to strings, complying only to the condition that $|\{x : K_f(x) = k\}| \leq 2^k$, where k-bit strings are used as the description of strings x for which $K_f(x) = k$. Of course, this would allow us to trivially and arbitrarily find, for a certain x, a f such that $K_f(x) = 0$ (by mapping the empty string to it)!

Consider now r distinct specification methods, $f_1, f_2, \ldots, f_r$. It is easy to construct a new method, f, that assigns to each x a complexity $K_f(x)$ that exceeds only by about $\log r$ bits the minimum of $K_{f_1}(x), K_{f_2}(x), \ldots, K_{f_r}(x)$. The idea is to use the first $\log r$ bits of p to identify the method f_i and then the appropriate program for f_i. Hence, we can say that

$$K_f(x) \leq K_{f_i}(x) + c, \forall i,$$

i.e., that the description method f **minorizes** (additively) every description method f_i.

The development of the theory of algorithmic information (also known as **Kolmogorov complexity**) is made possible by the remarkable fact that the class of computable functions possesses a universal element that is additively optimal. Under this relatively natural restriction on the class of description methods (i.e., computability), we obtain a well-behaved hierarchy of complexities and hence a minimal element.

Definition 4.4 (Conditional algorithmic complexity)
Let $x, y, p \in \mathbb{N}$ (recall that this is the same as considering $x, y, p \in \{0,1\}^*$, according to the

[6] Without loss of generality, from now on we will refer to these objects as strings and also consider that they are binary, i.e., $x, p \in \{0,1\}^*$. Also, as referred before, we assume the usual string ordering, $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$ and the implicit one-to-one correspondence between binary strings and non-negative integers (for example, the natural number "5" has the same meaning as the string "10").

usual enumeration of strings). Any partial recursive function ϕ, together with p and y, such that $\phi(\langle y, p\rangle) = x$, is a description of x. The complexity K_ϕ of x conditioned to y is defined by

$$K_\phi(x|y) = \min\{|p| : \phi(\langle y, p\rangle) = x\},$$

and $K_\phi(x|y) = \infty$ if there are no such p. Then, p is a program to compute x by ϕ, given y.

The key point of this theory is not that the universal description method necessarily gives the shortest description in each case, but instead that no other description method can improve on it infinitely often by more than a fixed constant.

Definition 4.5 (Kolmogorov complexity)
Let us fix an additively optimal universal ϕ_0 (the reference function) and define the conditional Kolmogorov complexity $K(x|y) = K_{\phi_0}(x|y)$. Also, fix a particular Turing machine (the reference machine), U, that computes ϕ_0. The unconditional Kolmogorov complexity is defined by

$$K(x) = K(x|\epsilon).$$

Theorem 4.1
For every computable function, f, there exists a constant, c, such that $K(f(x)) \leq K(x) + c$, for every x such that $f(x)$ is defined.

Theorem 4.2 (Non-computability of $K(x)$)
The Kolmogorov complexity is not computable.

To show why the Kolmogorov complexity is not a computable function, consider the following pseudo-code, that calls a hypothetical function "`K`", that calculates the Kolmogorov complexity of the string in the nth position of the usual string ordering, $K(n)$:

```
print_complex_string(min_k)
    n = 0
    while(true)
        if(K(n) > min_k)
            print(string(n))
            exit
        else
            n++
```

Obviously, for a value of "`min_k`" greater than the total size of the programs involved, the program would print a string with Kolmogorov complexity greater than the size of the program used to generate it, which is a contradiction! Therefore, it is not possible to have a program that calculates the Kolmogorov complexity of arbitrary strings.

5 Data compression

The main purpose of a data compression algorithm is to reduce the size of the data necessary to represent a certain amount of information. In this chapter, we address some of the most important approaches to data compression, characterized by allowing the total recovery of the original data, i.e., without losing any information. These are know as **lossless data compression algorithms**.

5.1 Variable-length coding

5.1.1 Optimal codes

For a given source distribution p_i, we want to find the prefix-free code with the minimum expected length, i.e., we want to minimize

$$L = \sum p_i l_i,$$

over all integers $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality, i.e.,

$$\sum 2^{-l_i} \leq 1.$$

By relaxing the integer constraint on l_i, we obtain the optimal code lengths, l_i^*, using an optimization method[7], that gives

$$l_i^* = -\log p_i.$$

This noninteger choice of codeword lengths yields an average codeword length

$$L^* = \sum p_i l_i^* = -\sum p_i \log p_i = H.$$

Theorem 5.1
The average length L of any instantaneous binary code for a source distribution p_i is greater than or equal to the entropy H, i.e.,

$$L \geq H,$$

with equality iff $2^{-l_i} = p_i$.

Proof.

$$H - L = -\sum p_i \log p_i - \sum p_i l_i = \sum p_i \log 2^{-l_i} - \sum p_i \log p_i =$$

$$= \sum p_i \log \frac{2^{-l_i}}{p_i} \leq \log \left(\sum p_i \frac{2^{-l_i}}{p_i} \right) = 0,$$

where the inequality results from the use of Jensen's inequality. □

[7] Consider, for example, that $l_i = -\log q_i$. Using Lagrange multipliers to solve this contrained optimization problem, we have to find the gradient of $-\sum p_i \log q_i + \lambda(\sum q_i - 1)$.

As shown, the choice of word lengths $l_i = -\log p_i$ yields $L = H$. However, in practice, these values may not be integers. Therefore, we might want to construct a code by rounding up the $-\log p_i$ lengths, i.e.,

$$l_i = \lceil -\log p_i \rceil.$$

Note that these lengths satisfy the Kraft inequality, because

$$\sum 2^{-\lceil -\log p_i \rceil} \leq \sum 2^{-(-\log p_i)} = \sum p_i = 1.$$

Moreover, we can write

$$-\log p_i \leq l_i < -\log p_i + 1 \;\Rightarrow\; H \leq L < H + 1.$$

Note that the optimal code can only be better than this code.

Theorem 5.2
Let $l_1^*, l_2^*, \ldots, l_m^*$ be the optimal codeword lengths for a source distribution p_i and let L^* be the associated expected length of an optimal code, i.e., $L^* = \sum p_i l_i^*$. Then

$$H \leq L^* < H + 1.$$

Proof. We have seen that if $l_i = \lceil -\log p_i \rceil$ then

$$H \leq L = \sum p_i l_i < H + 1.$$

Since L^* is less than L and since $L^* \geq H$, we have the theorem. □

In this last theorem, we have an overhead of at most one bit, due to the fact that $-\log p_i$ is not always an integer. We can reduce the overhead per symbol by spreading it out over many symbols. Therefore, let us consider a system in which we send strings of n symbols at once. Consider that L_n is the expected codeword length per input symbol, i.e., if $l(x_1 x_2 \cdots x_n)$ is the length of the codeword associated with string $x_1 x_2 \cdots x_n$, where $x_i \in \Sigma$, then

$$L_n = \frac{1}{n} \sum p(x_1 x_2 \cdots x_n) l(x_1 x_2 \cdots x_n) = \frac{1}{n} E[l(X_1, X_2, \ldots, X_n)].$$

Theorem 5.3
The minimum expected codeword length per symbol satisfies

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n^* < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n}.$$

Moreover, if $X_1, X_2, \ldots, X_n$ is a stationary stochastic process,

$$L_n^* \to H(\mathcal{X}),$$

where $H(\mathcal{X})$ is the entropy rate of the process.

Particularly, if the process is i.i.d., then

$$H \leq L_n^* < H + \frac{1}{n}.$$

5.1.2 A counting argument

Let us consider that there exists an algorithm, $\mathcal{C}$, that is able to compress all messages of n bits (or more). There are 2^n messages of n bits. By hypothesis, $\mathcal{C}$ is able to reduce the size of each of the 2^n messages to less than n bits.

How many codewords can be formed with less than n bits? There are 2^{n-1} with $n-1$ bits, 2^{n-2} with $n-2$ bits, $2^{n-(n-1)} = 1$ bit, ..., $2^{n-n} = 1$ with $n-n = 0$(!) bits.

Therefore, the total number of codewords with less than n bits is

$$\sum_{i=0}^{n-1} 2^i = \frac{1-2^n}{1-2} = 2^n - 1.$$

Hence, at least, two of the messages should share the same codeword, meaning that it is impossible to have a one-to-one codeword assignment. This implies that it is impossible to have a (reversible) compression algorithm that is able to compress all messages. For some of them, it will not compress or even it will expand its length.

5.1.3 Huffman codes

Huffman codes are variable-length codes developed by David A. Huffman (1925–1999) (Huffman, 1952). These codes are optimum, in the sense that they minimize the average length of the encoded messages, among all possible variable-length codes.

They are prefix-free codes, i.e., none of the codewords has a prefix made of a shorter codeword. Recall that, for example, if a given codeword is represented by the binary string "100", then the binary string "10001" cannot belong to that code. As shown before, this property ensures unique and instantaneous decoding.

The Huffman procedure for building the codes is based on two observations regarding optimum prefix codes:

- In an optimum code, symbols that occur more frequently should have shorter codewords than symbols that occur less frequently.
- In and optimum code, the two symbols that occur less frequently should have the same length.

The Huffman procedure contains the additional restriction that:

- The codewords of the two lowest probability symbols differ only in the last bit.

Therefore, for constructing the Huffman codes, the symbols (tree nodes) with the smallest probabilities are combined first. The new node gets the sum of the probabilities of the two combined nodes. This procedure is repeated until having a single node with a probability equal to one. Finally, the codewords are obtained by associating 0's and 1's to the branches of the tree.

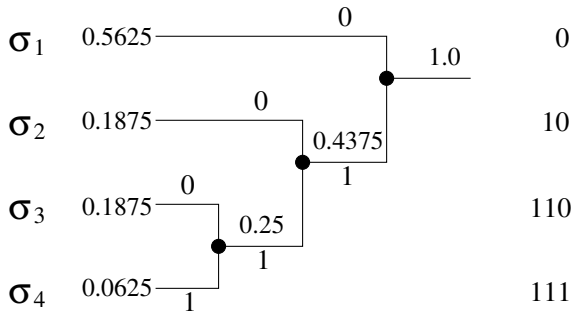


Figure 5.1: Example of tree associated with a 4-symbol Huffman code.

Example 5.1
*In the example of Fig. 5.1, the **mean number of bits/symbol** is*

$$1 \times 0.5625 + 2 \times 0.1875 + 3 \times 0.1875 + 3 \times 0.0625 \approx 1.69$$

The entropy of this information source is 1.62 *bits. Therefore, the **redundancy** of this code is* ≈ 0.07 *bits/symbol.*

The decoding process is straightforward. For example, suppose that the decoder received the binary string:

```
0100111101100
```

Then, the symbol sequence is

0	10	0	111	10	110	0
σ_1	σ_2	σ_1	σ_4	σ_2	σ_3	σ_1

When there are several nodes available for combining (i.e., with equal probabilities), it is possible to design different codes. In this case, which one shall we choose? For example, consider the following probability table, associated to a 5-symbol alphabet,

	σ_1	σ_2	σ_3	σ_4	σ_5
p_i	0.4	0.2	0.2	0.1	0.1

The first order entropy of this information source is 2.12 bits, and Fig. 5.2 shows two possible codes for this probability distribution. As can be verified, both codes have the same average number of bits/symbol:

$$2 \times 0.4 + 2 \times 0.2 + 2 \times 0.2 + 3 \times 0.1 + 3 \times 0.1 = 2.2$$

$$1 \times 0.4 + 3 \times 0.2 + 3 \times 0.2 + 3 \times 0.1 + 3 \times 0.1 = 2.2$$

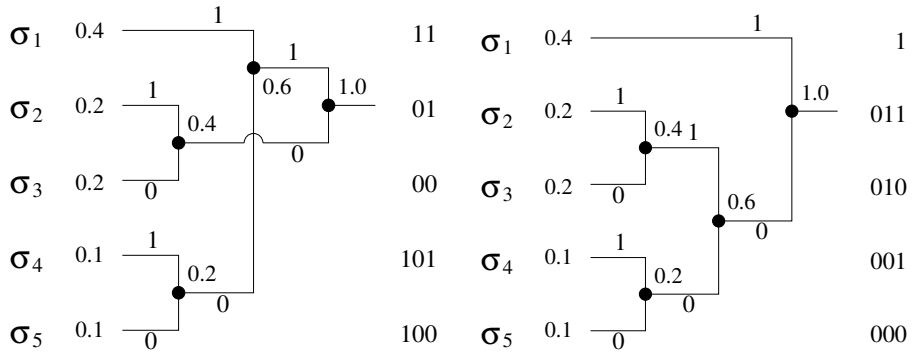


Figure 5.2: Two alternative Huffman codes for the same probability distribution.

After a sufficiently long transmission time, both codes are equivalent. However, those having higher variances might be less convenient, because they might require instantaneous higher bitrates and hence larger buffers. The first code has variance

$$0.4(2-2.2)^2 + 2 \times 0.2(2-2.2)^2 + 2 \times 0.1(3-2.2)^2 = 0.16,$$

whereas the second has variance 0.96.

Programming 5.1
Write a program that simulates the encoding process for a given probability distribution and corresponding Huffman code (for example, those shown in Fig. 5.2). Using that program, analyze the statistics of bitstream produced by the encoder with the aim of finding out if it is still compressible.

Programming 5.2
Using simulation, estimate the average time to overflow/underflow of the buffer, as a function of buffer size. As a possible example, verify that the leftmost code of Fig. 5.2 is less demanding than the code shown on the right, regarding buffer size.

5.1.4 Shannon-Fano codes

Shannon-Fano coding was proposed before Huffman coding and, although not optimum as Huffman coding, it nevertheless provides good variable-size codes. Shannon-Fano codes are like Huffman codes, in the sense that they also are variable-length and immediately decodable codes.

For constructing these codes, the symbols are first ordered by increasing or decreasing probability. Then, the symbols are split into two sets having as much as possible similar total probabilities. One of the sets gets a "0" label, whereas the other gets a "1" label. This is repeated until having sets with only one or two symbols.

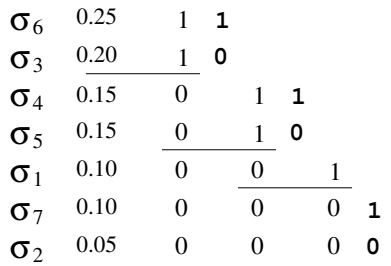


Figure 5.3: Example of construction of a Shannon-Fano code.

Example 5.2
Consider the following probability table:

σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0.10	0.05	0.20	0.15	0.15	0.25	0.10

The Shannon-Fano code for this probability distribution is shown in Fig. 5.3.

Note that whereas the Shannon-Fano tree is created from the root to the leaves, the Huffman algorithm works from leaves to the root.

5.1.5 Alphabet extension

When

$$p_i \to 2^{-k}, \quad k \in \mathbb{N}, \quad \forall i,$$

the efficiency of the variable-length codes gets closer to the entropy of the information source. Particularly, when the probabilities are powers of $1/2$, the average length of the codes equals the entropy of the source. In this case,

$$-\log p_i \in \mathbb{N}, \quad \forall i,$$

i.e., the optimum lengths of the codewords are integers. However, for small alphabets and for highly skewed probability distributions, the corresponding variable-length codes can be far from optimal!

Example 5.3
Let us consider two symbols, σ_1 and σ_2. Varying the values of p_1 and p_2, we get the following entropy values:

p_1	0.9	0.8	0.7	0.6	0.5
p_2	0.1	0.2	0.3	0.4	0.5
H	0.47	0.72	0.88	0.97	1.00

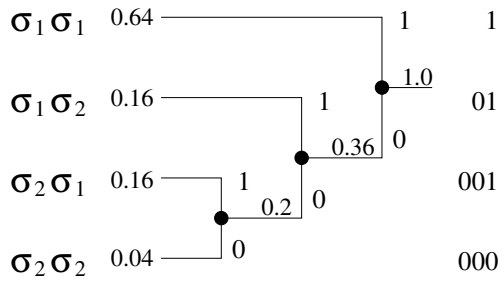


Figure 5.4: Huffman code resulting from the aggregation of pairs of symbols.

As expected, the larger the difference between these two probabilities, less bits, on average, are in theory needed for representing a message.

Consider, for example, the case where $p_1 = 0.8$ *and* $p_2 = 0.2$*:*

- *Using Huffman coding or Shannon-Fano coding, we get an average code length of one bit per symbol.*
- *This is 28% longer than what is theoretically necessary.*

Let us now consider groupings of pairs of symbols and assume ***independence****. The probabilities of the four combinations are*

Pair	Probability
$\sigma_1\sigma_1$	$0.8 \times 0.8 = 0.64$
$\sigma_1\sigma_2$	$0.8 \times 0.2 = 0.16$
$\sigma_2\sigma_1$	$0.2 \times 0.8 = 0.16$
$\sigma_2\sigma_2$	$0.2 \times 0.2 = 0.04$

Constructing a Huffman code for these new symbols (see Fig. 5.4), we get an average length of

$$1 \times 0.64 + 2 \times 0.16 + 3 \times 0.16 + 3 \times 0.04 = 1.56 \textit{ bits},$$

i.e., an average length of 0.78 *bits/original symbol.*

Consider now groupings of three symbols, with probabilities

Group	Probability
$\sigma_1\sigma_1\sigma_1$	$0.8 \times 0.8 \times 0.8 = 0.512$
$\sigma_1\sigma_1\sigma_2$	$0.8 \times 0.8 \times 0.2 = 0.128$
$\sigma_1\sigma_2\sigma_1$	$0.8 \times 0.2 \times 0.8 = 0.128$
$\sigma_1\sigma_2\sigma_2$	$0.8 \times 0.2 \times 0.2 = 0.032$
$\sigma_2\sigma_1\sigma_1$	$0.2 \times 0.8 \times 0.8 = 0.128$
$\sigma_2\sigma_1\sigma_2$	$0.2 \times 0.8 \times 0.2 = 0.032$
$\sigma_2\sigma_2\sigma_1$	$0.2 \times 0.2 \times 0.8 = 0.032$
$\sigma_2\sigma_2\sigma_2$	$0.2 \times 0.2 \times 0.2 = 0.008$

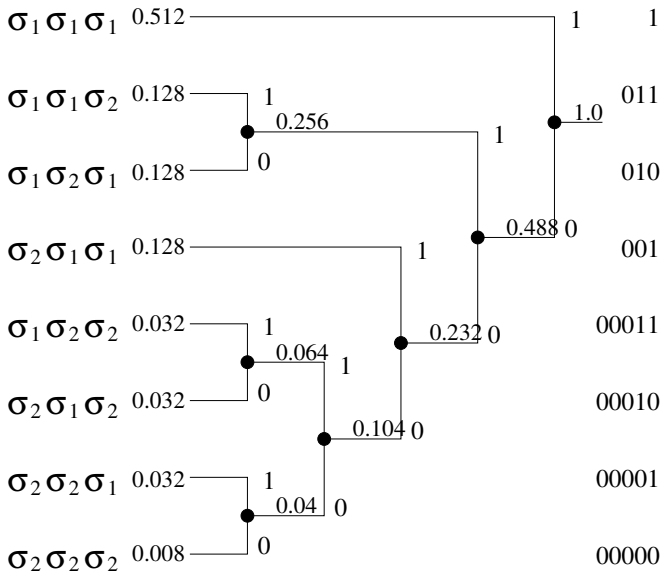


Figure 5.5: Huffman code for groups of three symbols.

The average length of the corresponding Huffman code is 2.184 *bits, i.e.,* 0.728 *bits/original symbol (see Fig. 5.5).*

5.1.6 Some other variable-length codes

Sometimes, it is necessary to design codes for representing, efficiently, integer numbers whose probabilities decrease with their values. In this case, it is possible to construct codes that are simpler than the Huffman codes, although sometimes with some loss of efficiency. On the other hand, these are "open" codes, i.e., they are able to accommodate rare, but possible, (large) numbers.

The **comma code** (also known as the **unary code**) is the simplest variable-length code. Each codeword is built using a string of 0's (1's), terminated by one 1 (0):

$$\begin{array}{lcl} 0 & - & 0 \\ 1 & - & 10 \\ 2 & - & 110 \\ 3 & - & 1110 \\ 4 & - & 11110 \\ \ldots & & \end{array}$$

Note that the decoder is just a counter.

Problem 5.1
Show that the unary code is optimum if the integers, i, occur with probabilities

$$p_i = \frac{1}{2^{(i+1)}}, \quad i = 0, 1, 2, \ldots$$

Shift-codes are generalizations of the comma code.

$B = 1$	$B = 2$	$B = 3$
0	00	000
10	01	001
110	10	010
1110	1100	011
11110	1101	100
111110	1110	101
1111110	111100	110
11111110	111101	111000
...	...	...

- These codes are organized by levels, based on a set of 2^B codewords of length B.
- One of those codewords is used to indicate the shift to the next level.
- For $B = 1$ we have the comma code.

Another type of shift code uses, besides the base, B, an increment, I. The first level is formed by all codewords of length B, except for the shift codeword. The next level (level 2) is formed by all codewords of $B + I$ bits, except for the shift codeword, concatenated with the (prefix) shift codeword of the previous level.

Generalizing, level L consists of all $B + I(L - 1)$ bit codewords, except for the shift sequence, concatenated with the prefix (shift codeword) of level $L - 1$.

Example 5.4

$B=1, I=1$		$B=2, I=1$		$B=1, I=2$	
L	*Codeword*	L	*Codeword*	L	*Codeword*
1	0	1	00	1	0
2	100	1	01	2	1000
2	101	1	10	2	1001
2	110	2	11000	2	1010
3	111000	2	11001	2	1011
3	111001	2	11010	2	1100
3	111010	2	11011	2	1101
3	111011	2	11100	2	1110
3	111100	2	11101	3	111100000
3	111101	2	11110	3	111100001
3	111110	3	111110000	3	111100010
	...		...		...

5.1.7 Golomb code

The **Golomb code** was proposed by Solomon W. Golomb (1932–2016) (Golomb, 1966). It relies on separating an integer into two parts:

- One of those parts is represented by a **unary code**.
- The other part is represented using a **binary code**.

In fact, the Golomb code is a family of codes that depend on an integer parameter, $m > 0$. The Golomb code is optimum for an information source following a distribution

$$p_i = \alpha^i(1-\alpha), \quad i = 0, 1, 2, \ldots$$

where

$$m = \left\lceil -\frac{1}{\log \alpha} \right\rceil .$$

An integer $i \geq 0$ is represented by two numbers, q and r, where

$$q = \left\lfloor \frac{i}{m} \right\rfloor \qquad \text{and} \qquad r = i - qm.$$

The quotient, q, can have the values $0, 1, 2, \ldots$, and is represented by the corresponding unary code. The remainder of the division, r, can have the values $0, 1, 2, \ldots, m-1$, and is represented by the corresponding binary code.

Example 5.5
Consider $i = 11$ *and* $m = 4$*. Then,*

$$q = \left\lfloor \frac{11}{4} \right\rfloor = 2, \quad r = 11 - 2 \times 4 = 3 \quad \rightarrow \quad 001\ 11$$

If m is not a power of 2, then the binary code (that represents the remainder of the division), is not efficient. In that case, the number of bits used can be reduced using a **truncated binary code**:

- First, define $b = \lceil \log m \rceil$.
- Encode the first $2^b - m$ values of r using the first $2^b - m$ binary codewords of $b - 1$ bits.
- Encode the remainder values of r by coding the number $r + 2^b - m$ in binary codewords of b bits.

Example 5.6
Consider $m = 5$*. In this case, we have* $b = \lceil \log 5 \rceil = 3$*. Therefore, values* $r = 0, 1, 2$ *are represented using 2 bits (respectively "00", "01" and "10"), whereas the values* $r = 3, 4$ *will be encoded using 3 bits (respectively "110" and "111").*

Example 5.7
Golomb code for $m = 5$ *and* $i = 0, 1, \ldots, 15$*:*

i	q	r	*Codeword*	i	q	r	*Codeword*
0	0	0	000	8	1	3	10110
1	0	1	001	9	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000

5.2 Dictionary based compression

In many cases, the information source produces recurring patterns. A possible approach to take advantage of this behaviour is to keep a **dictionary** with the (most frequent) patterns. When

one of those patterns occurs, it is encoded using a reference to the dictionary. If it does not appear in the dictionary, it can be encoded using some other (usually less efficient) method. Therefore, the idea is to split the patterns into two classes: frequent and infrequent.

Example 5.8
Suppose we have a source alphabet with 32 symbols (for example, 26 letters plus some punctuation marks). For an i.i.d. source, we would need 5 bits/symbol. Treating all 32^4 *(*$1\,048\,576$*) 4-symbol patterns as equally likely, it would imply 20 bits for each 4-symbol pattern.*

Now, suppose we put the 256 most common patterns in a dictionary. If the pattern is in the dictionary, send '0' followed by the (8-bit) index of the pattern in the dictionary. Otherwise, send '1' followed by the 20 bits encoding the pattern.

Notice that, if p *is the probability of finding the pattern in the dictionary, then the average code-length is*

$$L = 9p + 21(1 - p) = 21 - 12p.$$

To be useful, this scheme should attain $L < 20$*. This happens for* $p \geq 0.084$*. However, for an i.i.d. source, the probability of a 4-symbol pattern over a 32-symbol alphabet is only* $0.00025\ldots$

5.2.1 Tunstall codes

As we have seen, variable-length codes assign variable-length bit strings to the symbols of the alphabet. One of the problems with this approach is that errors in the encoded data propagate easily.

In the **Tunstall code**, all codewords are of equal length. Instead, each codeword represents variable-length groups of alphabet symbols.

Example 5.9
Consider the following 2-bit Tunstall code for the alphabet $\Sigma = \{A, B\}$*:*

Sequence	*Codeword*
AAA	*00*
AAB	*01*
AB	*10*
B	*11*

Then, the sequence "AAABAABAABAABAAA" is encoded as "001101010100".

The design of a code that has a fixed codeword length, but a variable number of symbols per codeword, needs to meet the following two conditions:

- We should be able to parse all source strings into substrings of symbols that appear in the codebook (dictionary). This property is usually known as "**generality**".
- We should maximize the average number of source symbols represented by each codeword.

Example 5.10
To understand the meaning of the first condition, consider now the following code:

Sequence	*Codeword*
AAA	*00*
ABA	*01*
AB	*10*
B	*11*

Let us try to encode the same sequence, "AAABAABAABAABAAA", but now using this new code:

- *First, we encode "AAA" with code "00".*
- *Then, we encode "B" with code "11".*
- *Next, we run into trouble, because there is no way to proceed...*

Lemma 5.1 (Dictionary generality)
A dictionary consisting of strings from an alphabet Σ satisfies the generality condition iff for every possible string of symbols x from Σ there exists at least one prefix of x in the dictionary.

Discussion topic 5.1
Discuss conditions that a dictionary must meet in order to obey the generality restriction.

A n-bit Tunstall code for an i.i.d. source over a size-m alphabet can be built using the following algorithm:

Algorithm 5.1 (Tunstall code)

- *Start with the m symbols of the alphabet in the codebook.*
- *Then, remove the entry with highest probability and add the m strings obtained by concatenating this symbol with every symbol of the alphabet. The new probabilities are the product (because independence is assumed) of the probabilities of the concatenated symbols—notice that this operation increases the size of the codebook from m to $m + (m - 1)$.*
- *Repeat this procedure k times, subject to the constraint $m + k(m - 1) \leq 2^n$, where k is the largest possible integer that verifies the condition.*

Example 5.11
Consider the design of a 3-bit Tunstall code for the alphabet

Symbol	*Probability*
A	*0.60*
B	*0.30*
C	*0.10*

Because "A" is the most probable symbol, we first obtain

Sequence	*Probability*
B	*0.30*
C	*0.10*
AA	*0.36*
AB	*0.18*
AC	*0.06*

Finally, we have

Sequence	*Probability*	*Code*
B	*0.300*	*000*
C	*0.100*	*001*
AB	*0.180*	*010*
AC	*0.060*	*011*
AAA	*0.216*	*100*
AAB	*0.108*	*101*
AAC	*0.036*	*110*

Discussion topic 5.2
Are the Tunstall codes uniquely decodable? Why?

5.2.2 String parsing

The ultimate objective of dictionary-based data compression is to parse the source string into substrings (also known as phrases or factors) whose corresponding codewords have the smallest possible total length. Parsing, i.e., creating a partition of the string, plays an important role in this process.

For an arbitrary dictionary, complying with the generality constraint, there are optimal (in the sense of producing the least number of phrases) ways of parsing the string to be compressed. Typically, they involve more than one pass over the string of data, using dynamic programming. More direct approaches, although possibly sub-optimal, rely on greedy strategies. In this case, assuming that processing is done left-to-right, the greedy algorithm chooses the longest possible prefix of the yet uncompressed string that exists in the dictionary.

For some special cases, the greedy strategy can be, in fact, optimal. This happens when the dictionaries are prefix-complete or suffix-complete. A dictionary is **prefix-complete** if for every string in the dictionary, all of its prefixes are also in the dictionary. A dictionary is **suffix-complete** if for every string in the dictionary, all of its suffixes are also in the dictionary.

Theorem 5.4 (Cohn and Khazan 1996)
With respect to a suffix-complete dictionary, greedy parsing left-to-right is optimal; dually, with respect to a prefix-complete dictionary, greedy parsing right-to-left is optimal.

5.2.3 LZ77 compression

In the 1970's, Jacob Ziv (1931–) and Abraham Lempel (1936–) proposed two different approaches for data compression, using dictionary-based principles. Contrarily to the Tunstall code that we have just seen, where the dictionary is built beforehand (i.e., it is **static**), these methods are **adaptive**—the dictionary is "learned" at the same time compression is performed.

The first algorithm, usually known as LZ77 (or LZ1), was presented in

- J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory (Ziv and Lempel, 1977).

Essentially, it relies on a separation of the data in two parts:

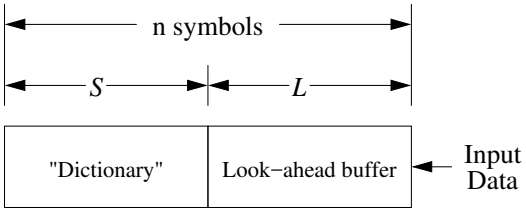


Figure 5.6: Separation between the "dictionary" and data to be encoded in LZ77.

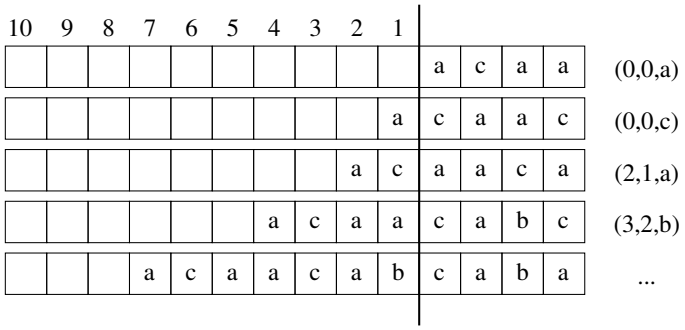


Figure 5.7: Simple example of LZ77 encoding.

- Data already encoded;
- Data that are to be encoded,

as shown in Fig. 5.6.

During operation, the data go first through a **look-ahead buffer** and then through a **search buffer** (the "dictionary"). The algorithm searches, in the "dictionary", for the largest possible prefix of the string in the look-ahead buffer. Obviously, the larger the prefix, the higher will be the coding efficiency.

The codeword that is generated in each coding step is composed of **three components**:

1. A pointer, that indicates the position of the matching string in the "dictionary".
2. The length of the string matched.
3. The first symbol that in the look-ahead buffer follows the matching string.

Figure 5.7 shows a simple example of encoding. The look-ahead buffer can also be used as a "dictionary" extension, as exemplified in Fig. 5.8.

In LZ77 encoding, a codeword requires

$$\lceil \log_2 S \rceil + \lceil \log_2 (L-1) \rceil + \lceil \log_2 m \rceil$$

bits, where S is the size of the "dictionary", L is the size of the look-ahead buffer and m is the size of the alphabet. Taking as example $S = 2048$, $L = 33$ and $m = 256$, a codeword would

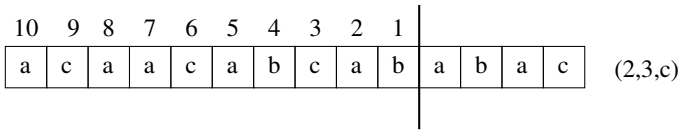


Figure 5.8: Example of the use of the look-ahead buffer for "dictionary" extension in LZ77 encoding.

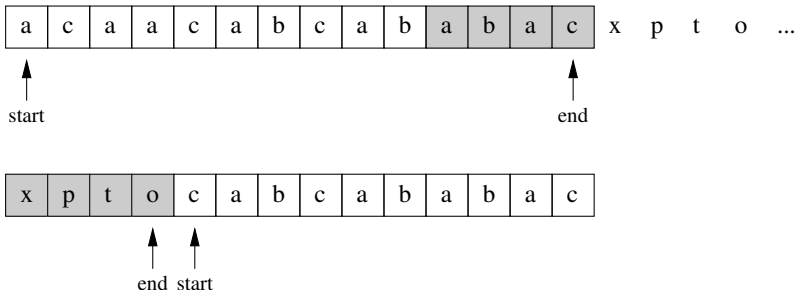


Figure 5.9: Look-ahead buffer implemented using a circular queue.

require $11 + 5 + 8 = 24$ bits, which is three times the number of bits required to represent a single symbol (8 bits).

Discussion topic 5.3
There is a number of practical questions that need to be addressed when implementing this compression algorithm, such as

- *The search time required by each coding step as a function of the dictionary size.*
- *The probability of finding matching sequences as a function of the dictionary size.*
- *The processing time required by each coding step as a function of the look-ahead buffer size.*
- *The probability of finding matching sequences as a function of the look-ahead size.*
- *The impact in codeword length when the sizes of the dictionary and/or look-ahead buffer are changed.*
- *The potential inefficiency of using three component codewords when having to encode isolated symbols.*

Discuss them.

The implementation of LZ77-type algorithms needs also to take into consideration the efficient use of data structures. For example, the use of circular buffers (see Fig. 5.9) and tree structures (see Fig. 5.10) may dramatically improve processing speed.

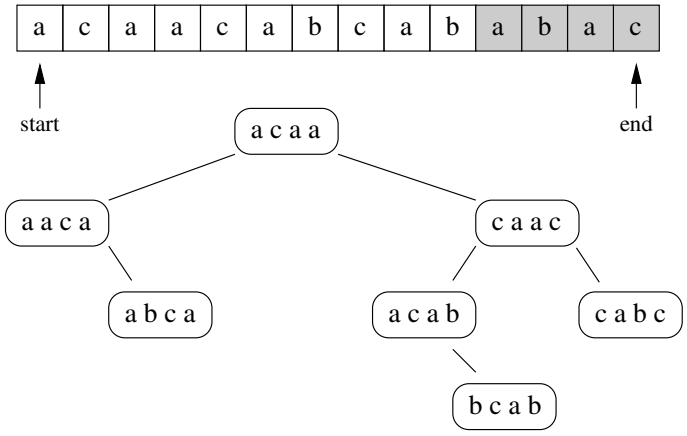


Figure 5.10: Dictionary implemented using a binary tree.

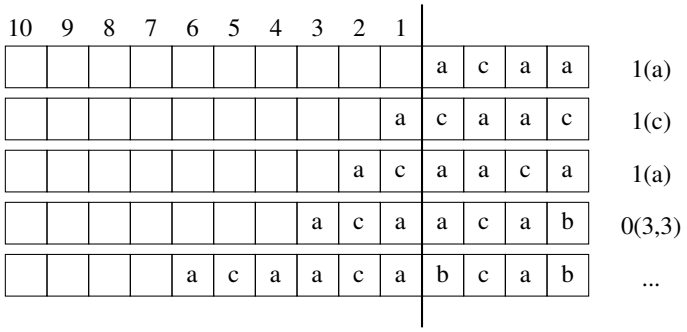


Figure 5.11: Example of LZSS encoding.

5.2.4 LZSS compression

LZSS is a variant of LZ77 proposed by James Storer (1953–) and Thomas Szymanski (Storer and Szymanski, 1982). As shown in the previous section, with LZ77 sometimes it is possible to produce codewords that use more bits than the original bits of the string it represents. With LZSS, this is avoided by enforcing codewords with only **two components**—one bit is used to indicate if the data are encoded or sent as uncoded, literal form. Therefore, when a sequence is not found in the dictionary, the symbol code is sent, instead of $(0, 0, \dots)$ as in LZ77. To distinguishing both cases, one bit is used. Figure 5.11 gives an example of an encoding using LZSS.

5.2.5 LZ78 compression

The second algorithm proposed by Jacob Ziv and Abraham Lempel, known as LZ78 (or LZ2), was published in

- J. Ziv and A. Lempel, *Compression of individual sequences via variable-rate coding*,

Dictionary:	a	aa	b	aaa	d	aab	aad	o
Index:	1	2	3	4	5	6	7	8
Codeword:	0,a	1,a	0,b	2,a	0,d	2,b	2,d	0,o

Figure 5.12: LZ78 encoding of string “aaabaaadaabaado”.

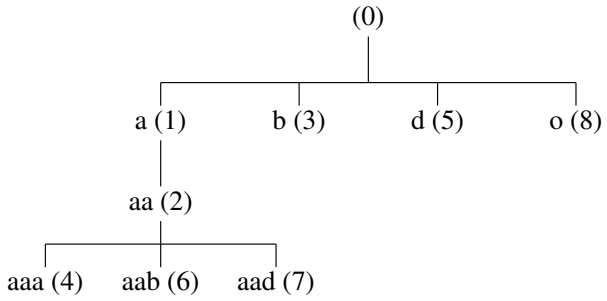


Figure 5.13: Tree data structure for fast access to the LZ78 dictionary.

IEEE Trans. on Information Theory (Ziv and Lempel, 1978).

LZ78 relies on a distinct approach from that of LZ77, where the use of an explicit dictionary is probability the most obvious. In LZ78 coding:

- The data are parsed into strings.
- Each string is built from the largest matching string found so far, plus the first non-matching symbol.
- The new string is represented using the index of the matching string (which is in the dictionary), followed by the code of the unmatched symbol.
- Finally, the new string is added to the dictionary.

Figure 5.12 shows how the string “aaabaaadaabaado” is represented by the LZ78 encoding algorithm. It is worth noting that, apparently, this new strategy seems not to put restrictions to the distance at which repeating patterns occur, whereas with LZ77, if this distance exceeds the size of the searching area (the “dictionary”), they are not taken into account. Nevertheless, it was shown that both techniques are asymptotically optimal for ergodic sources, i.e., asymptotically, its encoding efficiency approaches the entropy of the source.

As with LZ77-type algorithms, it is fundamental to use appropriate data structures to speed up processing. In the case of LZ78, a possibility is to represent the dictionary using tree structures, such as in the example of Fig. 5.13.

Discussion topic 5.4

In practice, the dictionary cannot grow without bound, because, for example,

- *The size of the codewords is related to the size of the dictionary.*
- *The amount of memory for storing the dictionary (both by the encoder and by the decoder) might be too large.*

Therefore, in practice, it is necessary to implement mechanisms for limiting the growth of the dictionary. Discuss advantages and disadvantages of possible mechanisms for doing so, such as,

- *Dictionary "freezing", when it reaches some predefined size.*
- *When the dictionary reaches some predefined size, it is deleted and started again (note that this is identical to block coding).*
- *When the dictionary is full, some entries are deleted, for example, those not used for a longer time or less used.*

Discussion topic 5.5
LZ77 relies on a suffix-complete dictionary, whereas LZ78 relies on a prefix-complete dictionary (explain why). Based on Theorem 5.4, discuss possible implications of this fact.

5.2.6 LZW compression

In 1984, Terry Welch (1939–1988) published a paper where solutions for some of the problems associated to LZ78 are proposed, namely (Welch, 1984):

- Symbols seen for the first time are encoded more efficiently (in LZ78, two-component codewords are generated in this case).
- The number of components of the codewords is reduced to just **one**.

Initially, all symbols of the alphabet are inserted in the dictionary. Then, the algorithm operates as follows:

- Symbols are read, one by one, from the string to encode.
- These symbols (σ) are concatenated to a string, x, while $x\sigma$ can be found in the dictionary.
- When it is not possible to add one more symbol, the index of x in the dictionary is sent, $x\sigma$ is inserted in the dictionary, and x is initialized with symbol σ.

Example 5.12

Consider the encoding of string "aaabaaadaabaado". Initially, the dictionary contains all symbols of the alphabet:

0	…	97	98	99	100	…	255
nul	…	a	b	c	d	…	

String: aa *abaaadaabaado; Codeword sent: 97*

…	97	98	99	100	…	256
…	a	b	c	d	…	aa

String: a aab *aaadaabaado; Codeword sent: 256*

…	97	98	99	100	…	256	257
…	a	b	c	d	…	aa	aab

String: aaa ba *adaabaado; Codeword sent: 98*

…	97	98	99	100	…	256	257	258
…	a	b	c	d	…	aa	aab	ba

String: aaab aaa *daabaado; Codeword sent: 256*

…	97	98	99	100	…	256	257	258	259
…	a	b	c	d	…	aa	aab	ba	aaa

String: aaabaa ad *aabaado; Codeword sent: 97*

…	97	98	99	100	…	256	257	258	259	260
…	a	b	c	d	…	aa	aab	ba	aaa	ad

String: aaabaaa da *abaado; Codeword sent: 100*

…	97	98	99	100	…	256	257	258	259	260	261
…	a	b	c	d	…	aa	aab	ba	aaa	ad	da

String: aaabaaad aaba *ado; Codeword sent: 257*

…	97	98	99	100	…	256	257	258	259	260	261	262
…	a	b	c	d	…	aa	aab	ba	aaa	ad	da	aaba

String: aaabaaadaab aad *o; Codeword sent: 256*

…	97	98	99	100	…	256	257	258	259	260	261	262	263
…	a	b	c	d	…	aa	aab	ba	aaa	ad	da	aaba	aad

...

Example 5.13

Decoding is performed as follows. Initially, the dictionary contains all symbols of the alphabet.

0	...	97	98	99	100	...	255
nul	...	a	b	c	d	...	

Codeword received: 97 ; String: a?

...	97	98	99	100	...	256
...	a	b	c	d	...	a?

Codeword received: 256 ; String: a aa?

...	97	98	99	100	...	256	257
...	a	b	c	d	...	aa	aa?

Codeword received: 98 ; String: aaa b?

...	97	98	99	100	...	256	257	258
...	a	b	c	d	...	aa	aab	b?

Codeword received: 256 ; String: aaab aa?

...	97	98	99	100	...	256	257	258	259
...	a	b	c	d	...	aa	aab	ba	aa?

Codeword received: 97 ; String: aaabaa a?

...	97	98	99	100	...	256	257	258	259	260
...	a	b	c	d	...	aa	aab	ba	aaa	a?

Codeword received: 100 ; String: aaabaaa d?

...	97	98	99	100	...	256	257	258	259	260	261
...	a	b	c	d	...	aa	aab	ba	aaa	ad	d?

Codeword received: 257 ; String: aaabaaad aab?

...	97	98	99	100	...	256	257	258	259	260	261	262
...	a	b	c	d	...	aa	aab	ba	aaa	ad	da	aab?

Codeword received: 256 ; String: aaabaaadaab aa?

...	97	98	99	100	...	256	257	258	259	260	261	262	263
...	a	b	c	d	...	aa	aab	ba	aaa	ad	da	aaba	aa?

...

5.3 Arithmetic coding

5.3.1 Motivation

We have seen that one of the main limitations of variable length codes is that they are optimum only if the probabilities of the symbols are powers of $1/2$.

Example 5.14
Suppose we have

$$p_1 = \frac{1}{2}, \quad p_2 = \frac{1}{4}, \quad p_3 = \frac{1}{4}$$

The first-order entropy of this information source is 1.5 *bits/symbol. Using a Huffman code, we have, for example,*

$$\sigma_1 \to 0 \quad \sigma_2 \to 10 \quad \sigma_3 \to 11$$

The average length of this code is 1.5 *bits/symbol and, therefore, its redundancy is zero (i.e., this code is optimal for this source).*

Example 5.15
Consider now that the probability distribution of the source symbols is

$$p_1 = \frac{2}{3}, \quad p_2 = \frac{1}{6}, \quad p_3 = \frac{1}{6}.$$

In this case, the entropy of the source is ≈ 1.26*. However, as in the previous case, the Huffman encoding algorithm assigns 1 bit to* σ_1 *and 2 bits to the other two symbols. The average length of this code is* ≈ 1.33*, i.e.,* 0.07 *bits per symbol more than the source entropy. The difference between the average code length and the entropy increases when* $\max(p_i) > 0.5$ *and* $\max(p_i) \to 1$.

As with Huffman coding, arithmetic coding needs an estimate of the probabilities of the symbols of the alphabet. Arithmetic coding represents the entire message by a single codeword: a number in the $[0, 1)$ interval. This is why arithmetic coding is not affected by the skewness of the probability distribution, as variable length codes might be. Besides, arithmetic coding has the advantage of allowing a clear separation between the **coding** process and source **modeling**.

5.3.2 Encoding

For encoding, the $[0, 1)$ interval is partitioned according to the probability distribution of the symbols. The only restriction is that the decoder has to use the same partitioning.

Example 5.16
Consider the following alphabet and corresponding probability distribution:

Symbol	*Prob.*	*Interval*
d	0.1	$[0, 0.1)$
e	0.3	$[0.1, 0.4)$
f	0.1	$[0.4, 0.5)$
g	0.05	$[0.5, 0.55)$
n	0.1	$[0.55, 0.65)$
r	0.2	$[0.65, 0.85)$
u	0.05	$[0.85, 0.9)$
space	0.1	$[0.9, 1)$

Let us consider the encoding of the message "fred".

*Initially, the **coding interval** is* $[0, 1)$*. The first symbol to encode, "f", has the* $[0.4, 0.5)$ *interval assigned to it, according to the probability table. After encoding this symbol, the coding interval will be* $[0.4, 0.5)$*. The next symbol, "r", is associated with the* $[0.65, 0.85)$ *interval. Then, the **lower limit** of the new coding interval will be*

$$0.4 + 0.65(0.5 - 0.4) = 0.465$$

*and the **upper limit***

$$0.4 + 0.85(0.5 - 0.4) = 0.485.$$

The next symbol, "e", transforms the current coding interval, $[0.465, 0.485)$*, into* $[0.467, 0.473)$*, according to*

$$0.467 = 0.465 + 0.1(0.485 - 0.465)$$

and

$$0.473 = 0.465 + 0.4(0.485 - 0.465).$$

After encoding symbol "d", we get the interval $[0.467, 0.4676)$.

Therefore, if the current coding interval is $[L^n, H^n)$, and if the next symbol to encode is represented by the interval $[l, h)$, then

$$L^{n+1} = L^n + l(H^n - L^n)$$

and

$$H^{n+1} = L^n + h(H^n - L^n).$$

Example 5.17
Consider the encoding of the binary string `0110`*, using* $P(0) = 0.6$ *and* $P(1) = 0.4$*. The encoded message is a number in the interval* $[0.504, 0.5616)$ *(see Fig. 5.14). Another possible view of the encoding process can be seen in Fig. 5.15, in this case with renormalized intervals.*

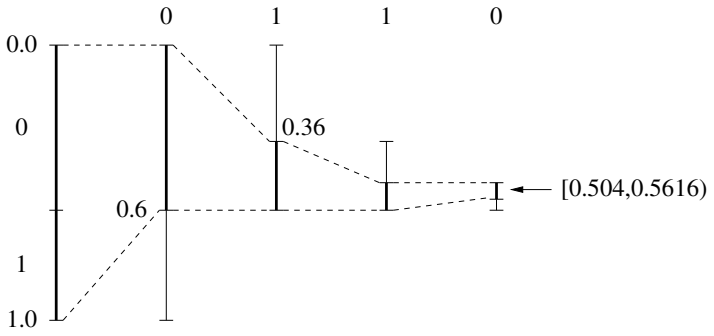


Figure 5.14: Example of the arithmetic encoding of a simple string.

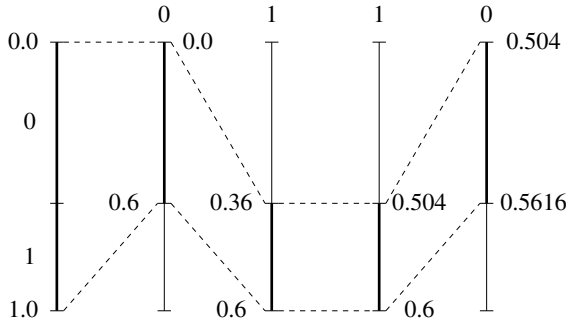


Figure 5.15: The same example as of Fig. 5.14, but with a different visualization of the intervals.

5.3.3 Decoding

For decoding the "fred" message, we proceed as follows. Let x be the value received by the decoder (i.e., the encoded message), such that $x \in [0.467, 0.4676)$. For example, consider $x = 0.467$. Since $x \in [0.4, 0.5)$, then the first symbol of the message is "f". The "effect" of this symbol is then eliminated from the encoded message, using the following procedure:

1. Subtract the lower limit of the "f" interval from x, i.e., $x - 0.4 = 0.467 - 0.4 = 0.067$.
2. Divide this new value by the range of the "f" interval (0.1, in this case): $0.067/0.1 = 0.67$.

The new number, $x = 0.67$, is in the $[0.65, 0.85)$ interval, meaning that "r" is the next symbol. Re-normalizing, we obtain $x = (0.67 - 0.65)/(0.85 - 0.65) = 0.1$. Because $x \in [0.1, 0.4)$, then the next symbol is "e". Re-normalizing again, $x = (0.1 - 0.1)/(0.4 - 0.1) = 0$. Since $x \in [0, 0.1)$, then the symbol is "d".

Notice that, there is no way to detect, automatically, the end of a message encoded using arithmetic coding. Therefore, it is necessary to indicate the length of the message (i.e., the number of symbols of the message) or to use a special "end-of-message" symbol.

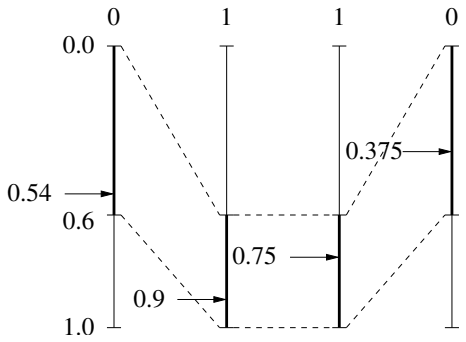


Figure 5.16: Decoding of the example depicted in Figs. 5.14 and 5.15.

Example 5.18
The decoding process of the previous example can be seen in Fig. 5.16, where $x = 0.54$ was considered.

5.3.4 Implementation issues

Obviously, it is not possible to obtain an useful encoder by direct implementation of the described process. In fact, the precision of the floating point representation of the computer would rapidly be exhausted. Nevertheless, even if we had access to a computer with infinite precision, it certainly would not be efficient to perform arithmetic operations with several thousands or millions of decimal places. Moreover, it is generally not acceptable to have to wait for the end of the message in order to start receiving the first bits of the encoded message.

As we saw earlier, as new symbols arrive at the encoder, the limits of the coding interval get closer to each other, implying the need of an increasing precision in order to distinguish them apart. However, it is reasonable to admit that, at a certain point, both limits will lay above 0.5 or below 0.5. When this happens, the most significant bit of L and H will stay permanently equal to 0, if $L < 0.5$ and $H < 0.5$, or equal to 1, if $L \geq 0.5$ and $H \geq 0.5$. Notice that

$$\begin{aligned} x \geq 0.5_{(10)} &\longrightarrow x = 0.1\ldots_{(2)} \\ x < 0.5_{(10)} &\longrightarrow x = 0.0\ldots_{(2)} \end{aligned}$$

Therefore, this bit can be sent out by the encoder.

On the other hand, we can expand the interval in the following way,

$$\begin{array}{lll} \text{Sent bit : } 0 & [0, 0.5) \longrightarrow [0, 1) & x \to 2x \\ \text{Sent bit : } 1 & [0.5, 1) \longrightarrow [0, 1) & x \to 2(x - 0.5), \end{array}$$

which, in principle, allow us to keep the precision under control.

But if $L \to 0.0111\ldots_{(2)}$ and $H \to 0.1000\ldots_{(2)}$?

To take into account this situation, whenever the coding interval belong to $[0.25, 0.75)$, we perform the expansion

$$[0.25, 0.75) \longrightarrow [0, 1) \qquad x \to 2(x - 0.25),$$

and a counter takes note of how many times these re-scalings occurred since the last bit has been sent by the encoder.

Notice that this operation is performed when $0.25 \leq L < 0.5$ ($L = 0.01\ldots_{(2)}$) and $0.5 \leq H < 0.75$ ($H = 0.10\ldots_{(2)}$). After this change of scale, we get $L = 0.0\ldots_{(2)}$ and $H = 0.1\ldots_{(2)}$. The result of this operation is that the second most significant bit is "deleted".

If, after incorporating the next symbol, $L \geq 0.5$ and $H \geq 0.5$, then the corresponding "1" is sent, followed by as many "0" bits as the value of the counter. On the other hand, if $L < 0.5$ and $H < 0.5$, it is sent the corresponding "0" bit, followed by as many "1" bits as the value of the counter.

6 Some concepts of digital signal processing

6.1 Signals

In very general terms, a **signal** is a representation of a certain quantity that varies over time and/or space or other variables of interest. For example, the voice of a person can be represented as a time signal that measures changes in air pressure.

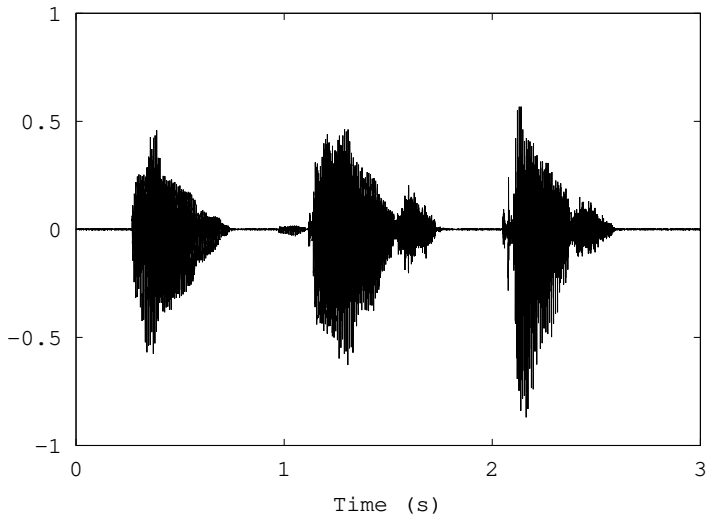


Figure 6.1: Audio signal corresponding to the portuguese words “um dois três” (“one two three”).

This pressure signal can be picked up by a microphone, converted into an electrical signal (audio signal), amplified and sent to a speaker, which transforms the audio back into sound signals. Fig. 6.1 shows a small example of an audio signal, with a duration of 3 seconds, corresponding to the portuguese words “um dois três” (“one two three”).

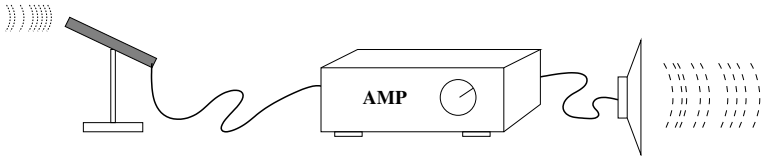


Figure 6.2: Sound capture, amplification and reproduction system.

Signals are transformed by **systems**. In the previous example, we can clearly identify three systems (Fig. 6.2): a microphone (converts audible variations of air pressure into electrical signals), an amplifier (increases the energy of the signal picked up by the microphone) and a speaker (converts the amplified audio signal into sound waves).

In Fig. 6.3, we can see another example of a system, as well as signals of various types, namely:

pressure, electrical, optical and electromagnetic. In fact, the concept of signal is so vast, that any quantity dependent on one or more variables can be considered a signal.

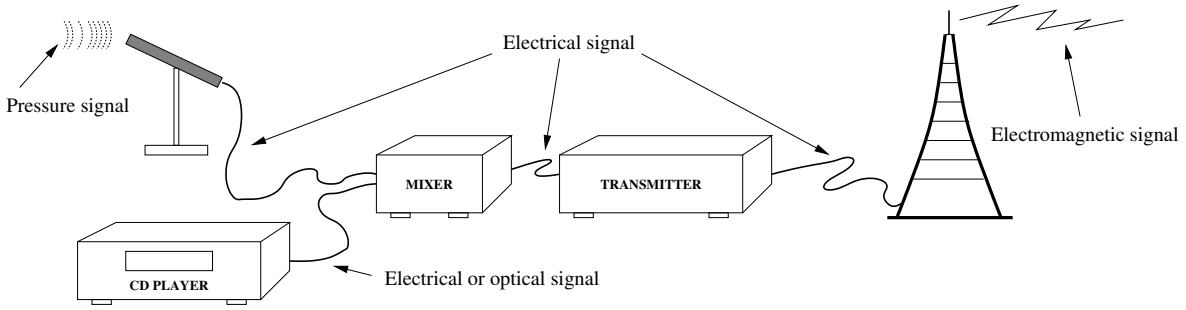


Figure 6.3: Audio broadcasting system.

Although a signal can depend on different variables, it is common to use the variable time when we want to represent a generic signal. In this case, the signal is represented, for example, by $x(t)$. Therefore, the signal shown in Fig. 6.1 could be represented, for mathematical manipulation purposes, by $x(t)$. So, we could say that the value of the signal at instant t_0 is $x(t = t_0)$ or, more simply, $x(t_0)$. Although this last notation can be confused with the representation of a complete signal, in general the context takes care to clarify its meaning.

A signal can have **scalar** or **vector**, **real** or **complex** values. In the example shown, the value of $x(t)$ could represent, for example, the electrical voltage at the output of the microphone, in which case we have $x(t) \in \mathbb{R}$. Since both t and $x(t)$ have values in $\mathbb{R}$, we say that $x(t)$ is an **analog** signal, that is, both the time variable and the signal value vary continuously. However, there are also signals that only have defined values at certain instants of time, t_i. These are called **discrete-time** signals. For example, the maximum daily temperatures over a year form a discrete-time signal, where the t_i are the days of the year. In this case, saying that $x(129)$ is equal to 34.8 corresponds to saying that on the 129th day of the year the maximum temperature was 34.8 degrees Centigrade.

A signal can also have **discrete values**. Take, for example, the number of people who are in a queue at a service counter. In this case, $x(t) \in \mathbb{N}_0$, that is, this signal has discrete values.

In summary, we have the following four combinations:

- **Signals continuous both in time and amplitude**
  $x(t) \in \mathbb{R}$ or $\mathbb{C}, t \in \mathbb{R}$
  Example: voltage at the output of a microphone
- **Signals continuous in time and discrete in amplitude**
  $x(t) \in \{\dots, x_{i-1}, x_i, x_{i+1}, \dots\} \subset \mathbb{R}$ or $\mathbb{C}, t \in \mathbb{R}$
  Example: number of people in a queue
- **Discrete-time and continuous-amplitude signals**
  $x(n) \in \mathbb{R}$ or $\mathbb{C}, n \in \mathbb{Z}$
  Example: maximum daily temperature

- **Signals discrete in time and amplitude**
  $x(n) \in \{\ldots, x_{i-1}, x_i, x_{i+1}, \ldots\} \subset \mathbb{R}$ or $\mathbb{C}, n \in \mathbb{Z}$
  Example: number of people served per hour by a service

Signals of the last type (discrete in time and amplitude) are particularly interesting to deal with in a computer. On the one hand, a computer has discrete memory cells, which can be used to store values of a signal at certain instants of time (discrete-time). On the other hand, due to the finite precision with which values can be stored in a computer, it is very convenient that the signal can only have values within a discrete and finite set (discrete-amplitude). Because signals discrete in time and amplitude are the most suitable for processing in a digital computer, we call them **digital signals**.

6.1.1 Some characteristics of the signals

In addition to the continuous-time / discrete-time and continuous-amplitude / discrete-amplitude characteristics, a signal can also be classified according to several other properties, including

- Whether it is deterministic or stochastic
- In the case of being deterministic, whether or not it is periodic
- In the case of being stochastic, whether or not it is stationary
- Whether it is finite-time or infinite-time

The **deterministic signals** are generated through algorithms that do not have randomness and, therefore, are predictable. For example, the signal $x(t) = \sin(t)$ is deterministic. This means that, for example, the graph of this signal for $0 \leq t \leq 4\pi$ is always the same, as shown in Fig. 6.4.

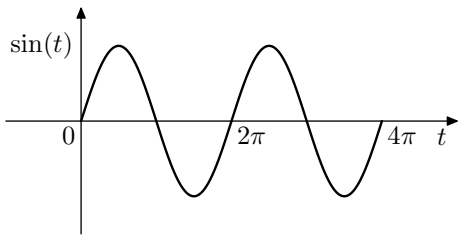


Figure 6.4: Example of a deterministic signal.

On the other hand, **stochastic signals** are generated by systems that have randomness, that is, whose value at a certain instant of time cannot be exactly determined *a priori*. The audio signal output from a microphone is an example of a stochastic signal. In fact, despite many

times a word is repeated in front of a microphone, the generated signal will always be different. Nevertheless, some features of these signals can be used, for example, to perform voice recognition.

A deterministic signal can be periodic, i.e., it can repeat itself exactly after a certain time interval, called the **period**. In the case of continuous-time signals, we have $x(t) = x(t+T), \forall t$, where T is the period. Obviously, if $x(t)$ is periodic with period T, then it is also periodic for periods kT with $k \in \mathbb{N}$. The smallest value of T, for which the relation $x(t) = x(t+T), \forall t$, is valid, is called the **fundamental period**. Unless otherwise indicated, whenever we refer to the period of a signal we are referring to its fundamental period. For discrete-time, the period is measured in number of samples, N, i.e., a signal $x(n)$ is periodic with period N if $x(n) = x(n+N), \forall n$.

A stochastic signal can be **stationary**, which means that its statistical characterization is maintained over time.[8] For example, this could indicate that the mean and variance of the signal remain constant over time. However, most interesting signals are not stationary, as they carry information. The human voice is one such example. Nevertheless, when analyzed in sufficiently small time intervals, we can often consider that stationarity is verified.

A signal is of **finite duration** if it starts at a certain point in time $t_i > -\infty$ and ends at another point in time, t_f, such that $t_i < t_f < +\infty$. In case this does not happen, we say that the signal is of **infinite duration**.

6.1.2 Analog-to-digital conversion

There are many signals that we would like to be able to process on a computer, but they are not digital signals. See the example of the audio signal picked up by a microphone. In this case, we can resort to the **analog-to-digital conversion** to transform continuous-time and continuous-amplitude signals into digital signals. Analog-to-digital conversion involves two operations: **sampling** and **quantization**.

The sampling operation is responsible for the passage from continuous-time to discrete-time. For this purpose, signal values are collected at certain time points. Typically, these instants of time are regularly spaced from T_s to T_s seconds. We call T_s the **sampling period** (Fig. 6.5), and $f_s = 1/T_s$ the **sampling frequency**. Therefore, when we say that a signal is sampled at 500 Hz, we are indicating that 500 samples are taken for every second of signal.

After the $x(t)$ signal has been sampled from T_s in T_s seconds, we have a sequence of values that correspond to the values of $x(t)$ at those instants of time, i.e., we have the sequence of values

$$\ldots, x(-2T_a), x(-T_a), x(0), x(T_a), x(2T_a), \ldots, x(nT_a), \ldots,$$

which we represent by $x(nT_s)$ or simply by $x(n)$.[9]

[8] There are strict ways to define stationarity, but they are beyond what we want to present here.

[9] By representing the sequence $x(nT_s)$ by $x(n)$ we are simplifying the notation, but also to introduce some loss of rigor. In fact, $x(n)$ designates any sequence of values, indexed by the integer variable n, which may or may not

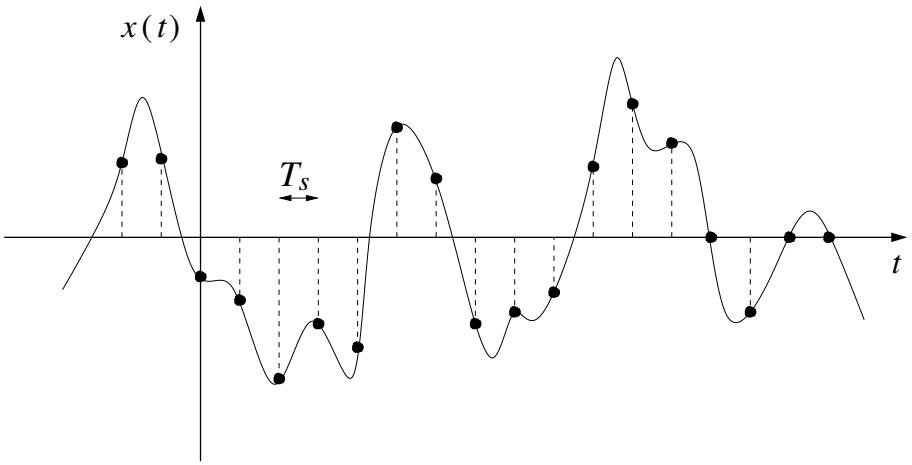


Figure 6.5: Signal sampling.

We will further show that, depending on certain properties of $x(t)$, it is possible to find values of T_s for which the sequence $x(nT_s)$ fully describes $x(t)$. In other words, in that case it will be possible to reconstruct exactly $x(t)$ based on the samples $x(nT_s)$.

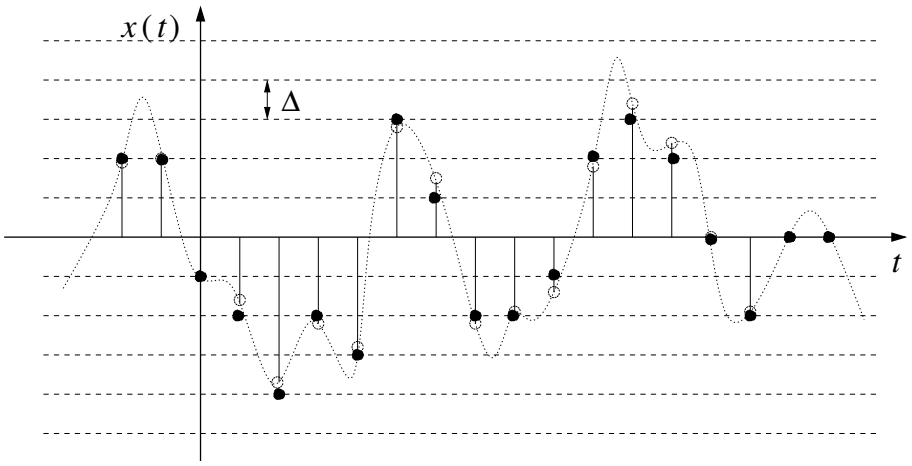


Figure 6.6: Signal quantization.

The quantization operation introduces errors (or **noise**, according to signal processing terminology) in the sample values. Unlike sampling, which is possible without introducing an error, it is not possible to do quantization without introducing irreversible errors. However, it is possible to calculate maximum values for the error made and reduce it as much as desired. One of the most common measures to express the introduced error is the **signal-to-noise ratio**, which

correspond to a temporal sequence and, even then, may not correspond to evenly spaced samples. However, we believe that the simplification of the notation justifies this inaccuracy, although whenever there may be doubt, we will opt for the complete notation.

is defined by the expression

$$\text{SNR} = 10 \log_{10} \frac{E_x}{E_r} \quad \text{dB (decibel)}, \tag{6.1}$$

where E_x and E_r indicate, respectively, the **energy** of the signal and of the noise, being the energy of a signal $x(t)$, E_x, calculated through

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 \, dt, \tag{6.2}$$

or

$$E_x = \sum_{n=-\infty}^{\infty} |x(n)|^2, \tag{6.3}$$

in the case of a discrete-time signal.

Note that, according to both (6.2) and (6.3), it may not be possible to calculate the energy of a certain signal. This happens if the respective integrals or summations do not converge. In fact, there are many signals of infinite energy, an obvious example being the class of periodic signals (what is the energy of the signal $\sin(t)$?). In this case, we can calculate the **power** of the signal, i.e., its energy per unit of time. The power of a signal is given by

$$P_x = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} |x(t)|^2 \, dt, \tag{6.4}$$

in the continuous-time case or

$$P_x = \lim_{N \to \infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x(n)|^2 \tag{6.5}$$

in the discrete-time case.

Note that, according to the definitions just presented, a finite-energy signal has zero power. These signals are called **energy signals**. On the other hand, a signal with non-zero power necessarily has infinite energy, and is called a **power signal**.

In the case of power signals, the signal-to-noise ratio defined in (6.1) changes to the form

$$\text{SNR} = 10 \log_{10} \frac{P_x}{P_r} \quad \text{dB}, \tag{6.6}$$

where P_x and P_r indicate, respectively, the power of the signal and of the noise.

The calculation of the energy according to (6.2) or (6.3), or of the power according to (6.4) or (6.5), applies to the cases where the signal is deterministic, or else it is an instance of a stochastic signal. However, we may also be interested in determining the "average" signal-to-noise ratio associated with a digitalization process of a certain class of stochastic signals. In

this case, we will have to use probabilistic tools and, in particular, the expected value (or mean value) of a random variable, X, which is given by

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)\, dx, \tag{6.7}$$

where $f_X(x)$ represents the probability density function of X.[10] Then, the power of a stochastic signal is obtained by calculating

$$P_x = E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)\, dx. \tag{6.8}$$

Let us then assume that a signal, $x(t)$, uniformly distributed over the interval $[-A/2, A/2]$,[11] is processed by a quantizer with 2^b levels. If the quantization levels are evenly spaced, then this spacing will be $\Delta = A/2^b$ and the maximum quantization error will be $r_{\max} = \Delta/2$ (Fig. 6.7 shows an example with 4 levels).

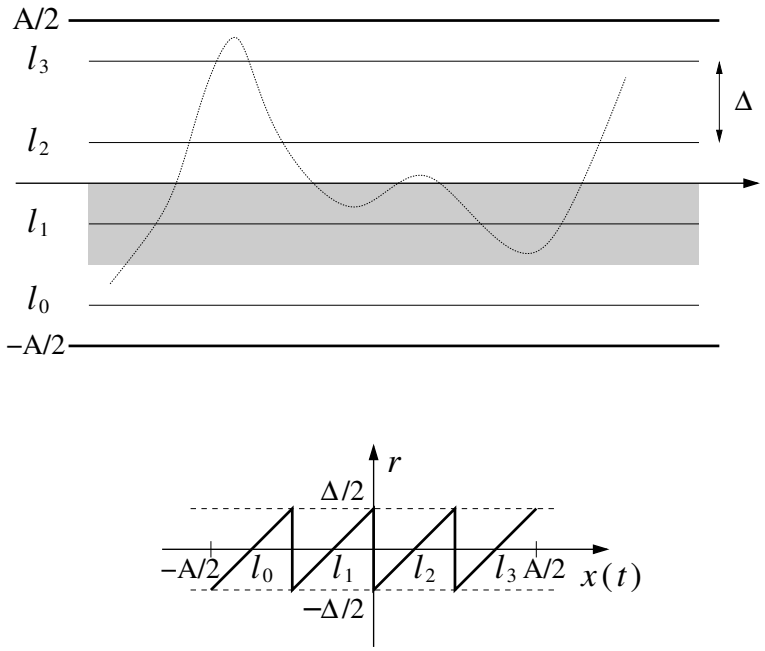


Figure 6.7: Example of quantization to four levels: l_0, l_1, l_2, l_3. Note that all signal values that occur within the shaded range will be represented by the value l_1.

Since $x(t)$ is uniformly distributed, we have

$$f_X(x) = \frac{1}{A}$$

[10] If a random variable X is characterized by a probability density function $f_X(x)$, then the probability that X has values between a and b is given by $\int_a^b f_X(x)\, dx$.

[11] We say that a random variable, X, is uniformly distributed over the interval $S_X = [-A/2, A/2]$, if the probability that it has values in a certain sub-interval of S_X of length $L \leq A$ is L/A.

and also

$$f_R(r) = \frac{1}{\Delta},$$

since the random variable associated with the quantization error also has a uniform distribution. So, the power of the quantization noise is

$$P_r = E[R^2] = \int_{-\Delta/2}^{\Delta/2} r^2 \frac{1}{\Delta} dr = \frac{\Delta^2}{12}$$

and the signal power is

$$P_x = E[X^2] = \int_{-A/2}^{A/2} x^2 \frac{1}{A} dx = \frac{A^2}{12}.$$

Since

$$\frac{A}{\Delta} = 2^b,$$

we have

$$\text{SNR} = 10 \log_{10} \frac{P_x}{P_r} = 10 \log_{10} \frac{A^2}{\Delta^2} = 20 \log_{10} 2^b \approx 6.02b \quad \text{dB}. \tag{6.9}$$

The relation (6.9) is well known and is generally used to estimate the signal-to-noise ratio obtained when representing each sample of the signal with b bits. As we can see, for each additional bit, the signal-to-noise ratio increases by approximately 6 dB. However, we cannot forget that this relation was derived assuming the uniform distribution (and zero mean) of the signal to be quantized, a characteristic that is often not verified.

Example 6.1

Let a signal, $x(t)$, be sampled at 1 kHz, which samples are to be stored in 6 bits. Considering that $x(t)$ has values between -1 and 1, we intend to determine an expression for the levels of a uniform quantizer, in order to minimize the maximum error. We also want to know what will be the generated binary data rate.

The spacing between quantizer levels is $\Delta = \frac{2}{2^6} = \frac{1}{2^5}$. Therefore, the 64 levels must be positioned at

$$l_k = -1 + \frac{\Delta}{2} + k\Delta = -1 + \frac{1}{2^6} + \frac{k}{2^5}, \quad k = 0, 1, \ldots, 63$$

so that the maximum error is minimized. In this case, the maximum error is $\frac{\Delta}{2} = \frac{1}{2^6}$.

Since 1000 samples are generated per second and each sample is stored in 6 bits, then the binary rate generated is $6 \times 1000 = 6$ kbps (kilobits per second).

Note that, generically, if the signal to quantize has a variation between $A_{\min}$ and $A_{\max}$, then the levels of the uniform quantizer are given by

$$l_k = A_{\min} + \frac{\Delta}{2} + k\Delta, \quad k = 0, 1, \ldots, 2^b - 1,$$

where

$$\Delta = \frac{A_{\max} - A_{\min}}{2^b}$$

and b is the number of bits used to represent each sample.

Example 6.2

The audio signal stored in a CD is sampled at 44.1 kHz, each sample being represented with 16 bits. It is intended to estimate the signal-to-noise ratio resulting from the uniform quantization process applied to the audio signal, and also calculate the bit rate at the output of a CD player.

If the audio signal were evenly distributed, we would have a signal-to-noise ratio equal to $16 \times 6 = 96$ dB. However, as in general the distribution of an audio signal is not uniform, this value should be understood merely as an estimate.

The binary rate at the output of the CD player is given by $44\,100 \times 16 \times 2 = 1\,411\,200$ bps (bits per second), i.e., approximately equal to 1.4 Mbps (megabits per second). Note that the "$\times 2$" factor results from the fact that CD audio is stereo and, therefore, there are two channels.

6.1.3 Multi-dimensional signals

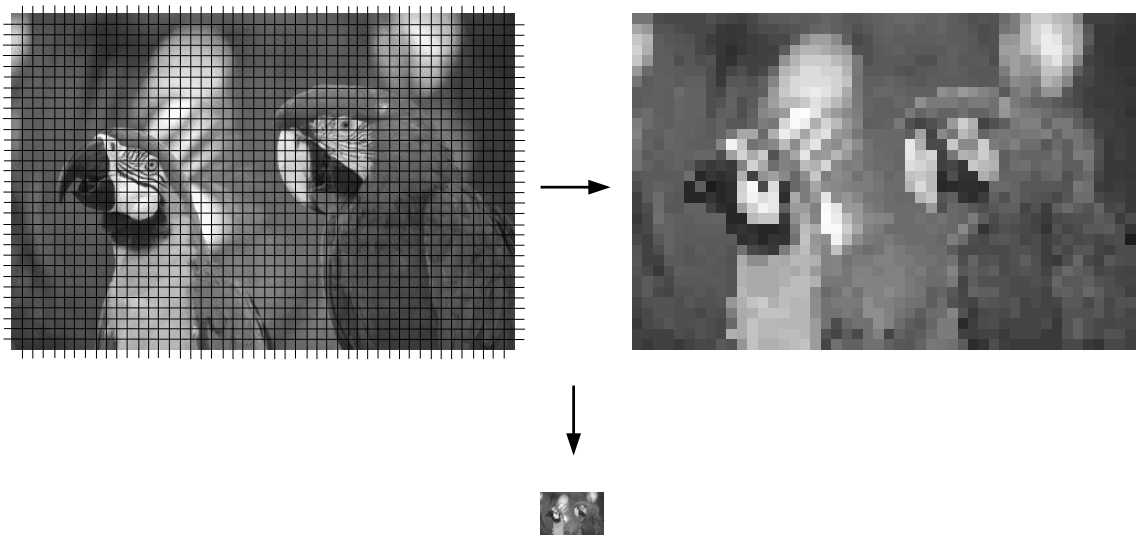
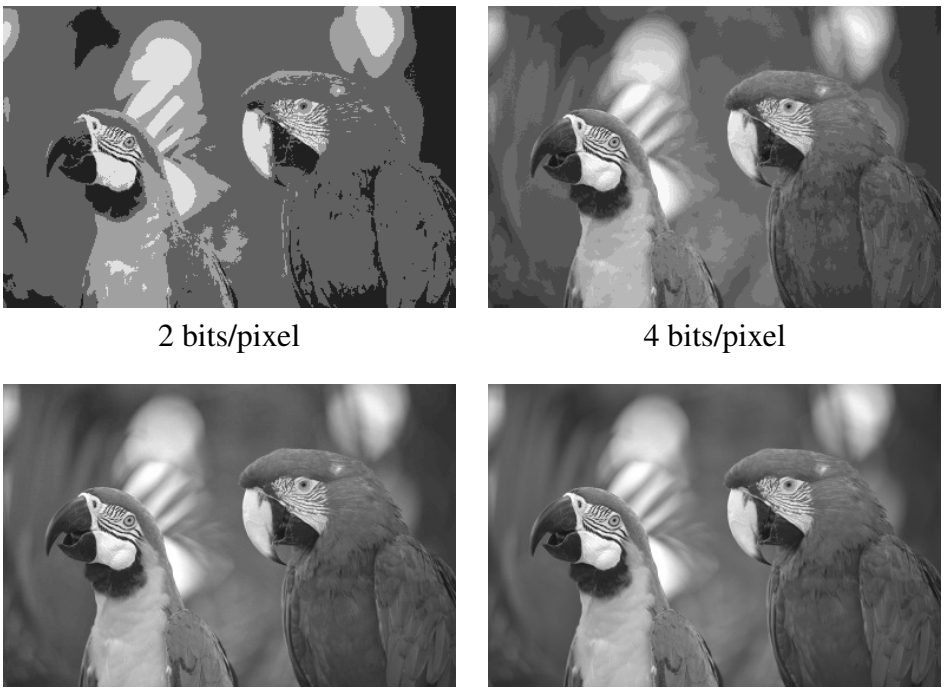


Figure 6.8: Image sampling.

We are daily confronted with information carried in the form of signals with two dimensions (images) and three dimensions (video). An image can be seen as a two-dimensional signal, $f(x, y)$, where x and y represent spatial coordinates. The value and meaning of $f(x, y)$ at a given point in space, (x, y), depends on the source responsible for producing the image. However, it is generally assumed that $f(x, y) \geq 0$ and that both (x, y) and $f(x, y)$ assume



2 bits/pixel 4 bits/pixel

6 bits/pixel 8 bits/pixel (original)

Figure 6.9: Quantization effects on images.

continuous values. Therefore, to convert $f(x, y)$ into a digital image, it is necessary, on the one hand, to perform a spatial sampling and, on the other hand, to quantize the amplitude of each resulting sample. Fig. 6.8 illustrates the process of sampling an image, whereas Fig. 6.9 shows the effect of quantization.

A digital image is represented by a rectangular matrix of scalars or vectors, $f(i, j)$, which we call picture elements or *pixels*. In general, we have $f(i, j) \in \mathcal{I} \subset \mathbb{N}_0^n$, i.e., each pixel is a non-negative integer or a vector of non-negative integers.

The most common types of images are (Fig. 6.11 presents an example of each of these four types):

- **Black and white or binary**
  $f(i, j) \in \{0, 1\}$
- **Gray level**
  $f(i, j) \in \{0, 1, \ldots, 2^b - 1\}$
- **Indexed color (indexes in a color map)**
  $f(i, j) \in \{0, 1, \ldots, 2^b - 1\} \xrightarrow{\alpha} \mathcal{I} \subset \{0, 1, \ldots, 2^{b'} - 1\}^3$
- **Color (for example, RGB)**
  $f(i, j) \in \{0, 1, \ldots, 2^b - 1\}^3$

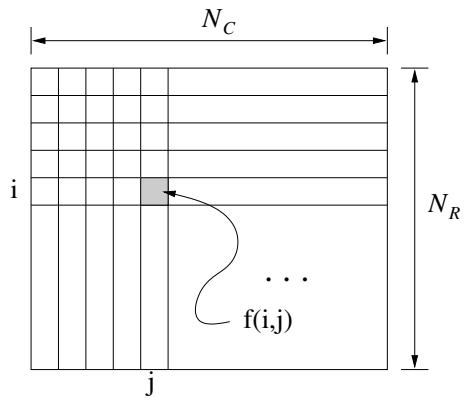


Figure 6.10: Digital image, where N_C and N_R respectively designate the number of columns and rows of the image.

To a temporal sequence of digital images we call **digital video**, which we represent by $v(i, j, n)$ or by $v(i, j, nT_s)$ if we want to evidence the sampling period over time. In this case, the period T_s is the time that elapses between the presentation of two consecutive video frames. The inverse of T_s is the **frame rate**.

For each value of n there corresponds a digital image, that is, we can write $f(i, j) = v(i, j, k), k =$ constant, which we denote by **frame** k of the video sequence v.

6.1.4 Some elementary signals

Although most real-life signals have a complex shape, simple signals are often used to study systems. There are several reasons for this approach, among them the ease of doing mathematical manipulations, the fact that some of these simple signals approximate some real-life signals relatively well, and probably the most important reason, because it is possible to represent complicated real-life signals by combining simple signals.

One of these elementary signals is the **unit step** (or Heaviside step function, Fig. 6.12), defined as

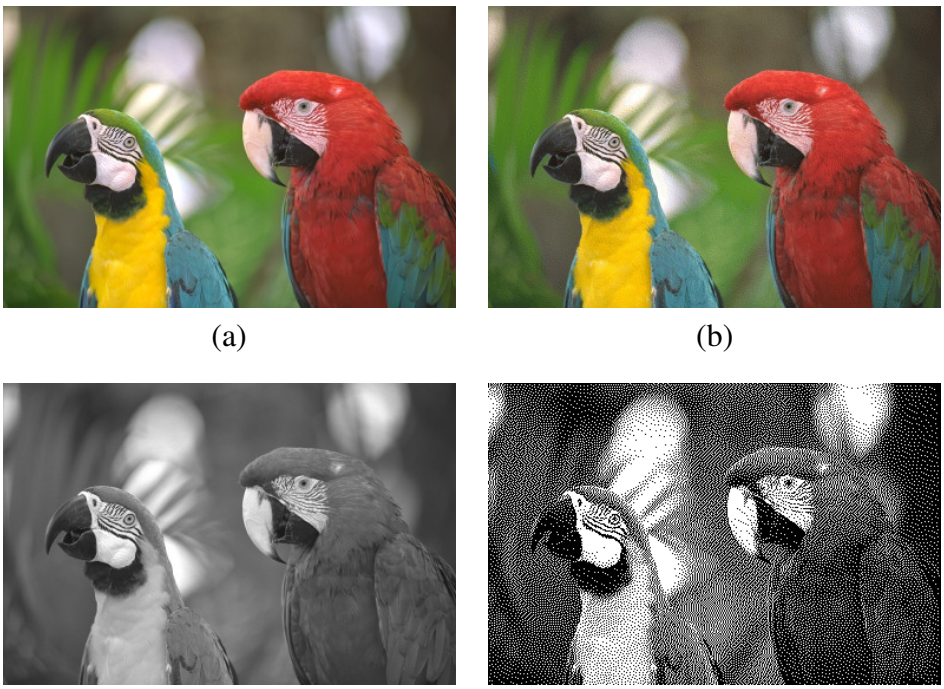
$$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t > 0 \end{cases} \tag{6.10}$$

or

$$u(n) = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases} \tag{6.11}$$

in the discrete-time case. Note that, although $u(t)$ is a continuous-time signal, this does not imply that it is continuous: $u(t)$ has a discontinuity at $t = 0$.

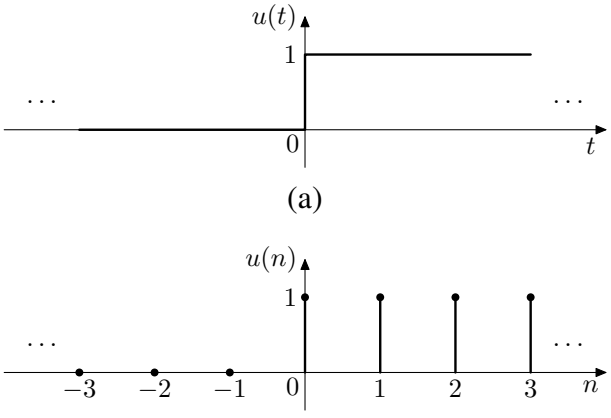
The **unit impulse** (Fig. 6.13) is another important elementary signal. In the case of continuous-



(a) (b)

(c) (d)

Figure 6.11: Examples of common digital images: (a) color; (b) indexed color (256 colors); (c) gray level (256 levels); (d) binary (2 levels).



(a)

(b)

Figure 6.12: Unit step: (a) continuous-time; (b) discrete-time.

time, this signal (also called Dirac delta) is defined by

$$\delta(t) = 0, \text{ for } t \neq 0 \tag{6.12a}$$

$$\delta(t) \text{ is undefined for } t = 0 \tag{6.12b}$$

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1. \tag{6.12c}$$

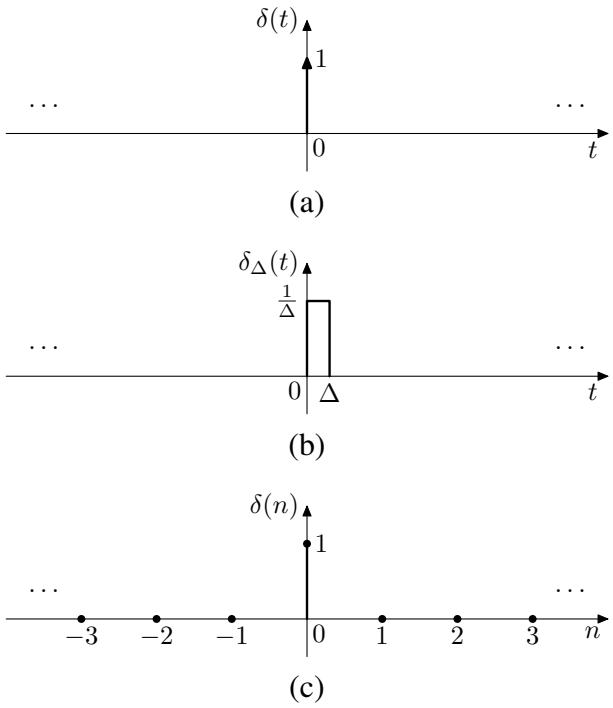


Figure 6.13: (a) Continuous-time unit impulse; (b) Example of a signal used to approximate the $\delta(t)$ signal; (c) Discrete-time unit impulse.

In fact, $\delta(t)$ is not a function in the mathematical sense of the term, but a distribution (or generalized function). Although the theory necessary to rigorously address this "function" is outside the scope of many introductory signal processing texts, it is nevertheless used given the ease with which it allows to explain certain results. In these situations, the Dirac delta is often seen as a limit case of another function, $\delta_\Delta(t)$, defined, for example, through

$$\delta_\Delta(t) = \begin{cases} \dfrac{1}{\Delta}, & 0 \leq t < \Delta \\ 0, & t < 0 \vee t > \Delta \end{cases} . \tag{6.13}$$

Notice that, according to (6.12),

$$\int_{-\infty}^{t} \delta(\tau)\, d\tau = u(t), \tag{6.14}$$

by which we might understand $\delta(t)$ as the "derivative" of $u(t)$.

The discrete-time case is much simpler, since the digital unit impulse is defined through

$$\delta(n) = \begin{cases} 0, & n \neq 0 \\ 1, & n = 0 \end{cases} . \tag{6.15}$$

The **exponential** signals,

$$x(t) = e^{\lambda t}, \quad \lambda \in \mathbb{R}, \tag{6.16}$$

are also of utmost importance. In the case the exponent is a real value, these signals show an always increasing or always decreasing behavior (except when the exponent is zero).

The **sinusoidal** signals are also very important in signal processing. These are related to the **complex exponentials**, since, by the Euler relation, we have

$$e^{\mathtt{j}\theta} = \cos\theta + \mathtt{j}\sin\theta, \tag{6.17}$$

where $\mathtt{j} = \sqrt{-1}$ is the imaginary unit.

From this result, two very important relations can be drawn, namely,

$$\cos\theta = \frac{e^{\mathtt{j}\theta} + e^{-\mathtt{j}\theta}}{2} \tag{6.18}$$

and

$$\sin\theta = \frac{e^{\mathtt{j}\theta} - e^{-\mathtt{j}\theta}}{2\mathtt{j}}. \tag{6.19}$$

In continuous-time, a sinusoidal signal is represented by

$$x(t) = \sin(2\pi f t) = \sin(\omega t), \tag{6.20}$$

where f and ω represent the sine frequency, respectively, in Hz and rad/s. Regarding the discrete-time case, we have

$$x(n) = \sin(2\pi f n T_s) = \sin\left(2\pi \frac{f}{f_s} n\right), \tag{6.21}$$

where T_s and f_s represent, respectively, the sampling period and the sampling frequency. Note that sampling a periodic signal with period T does not necessarily result in a periodic signal with the same period (in fact, the resulting signal may not be even periodic). Also note that

$$\sin\left(2\pi \frac{f}{f_s} n\right) = \sin\left(2\pi \frac{f + k f_s}{f_s} n\right), k \in \mathbb{N}_0. \tag{6.22}$$

(Show why and try to discover some implications of this property).

6.1.5 Operations on the independent variable

There are several operations in signal processing that can be described through modifications or transformations of the independent variable, t or n. Among them are time inversion, time delay and advancement, time compression and expansion.

Let us consider the signal

$$y(t) = x(-t). \tag{6.23}$$

The $y(t)$ signal consists of the **time inversion** of the $x(t)$ signal. This is the effect that happens, for example, when we play a recording in the opposite direction. The equivalent for the digital case is obvious: if the original signal is $x(n)$, then the time-inverted signal is $x(-n)$.

Let us now consider the signal

$$y(t) = x(t - \tau). \tag{6.24}$$

Note that the signal $y(t)$ has a value equal to $x(0)$ for $t = \tau$. Therefore, if $\tau > 0$, this value happens temporally after $t = 0$. Hence, the signal $y(t)$ is said to be a **delayed** version of the signal $x(t)$. In the case where $\tau < 0$, the signal $y(t)$ displays the value $x(0)$ before the instant $t = 0$, so in this situation the signal $y(t))$ is an **advanced** version of $x(t)$. Here, too, the adaptation to the discrete-time situation is obvious,

$$y(n) = x(n - k), \quad k \in \mathbb{Z}. \tag{6.25}$$

Consider now the signal

$$y(t) = x(\alpha t), \quad \alpha \in \mathbb{R}. \tag{6.26}$$

The $y(t)$ signal is a version of $x(t)$ that is "faster" (if $|\alpha| > 1$) or "slower" (if $|\alpha| < 1$), thus obtaining **temporal compression** in the first situation and **temporal expansion** in the second case. As an illustrative practical example of these operations, we can point out the case in which a recording is reproduced at double the speed (temporal compression) or at half the original speed (temporal expansion).

Any linear transformation of the independent variable can be expressed by combining the operations just described (time inversion, time expansion or compression, and delay or advance). For example, the signal $x(\alpha t - \tau)$ results from applying a delay (advance if τ is negative) of τ time units, followed by a compression (expansion if $|\alpha| < 1$) and even an inversion if $\alpha < 0$. Fig. 6.14 presents examples of the various operations on the independent variable, as well as combinations between them.

6.2 Systems

In the context of signal processing, a system is a device, process or algorithm that, given an input signal $x(t)$, produces an output signal $y(t)$. We express the action of a system, H, on a signal, $x(t)$, by writing

$$y(t) = H[x(t)]. \tag{6.27}$$

Fig. 6.15(a) presents a graphical representation of a generic system with N inputs and M outputs. Our attention will, however, go to systems with one input and one output, as is the case with the system illustrated in Fig. 6.15(b).

One of the simplest systems to describe mathematically is the **ideal amplifier**,

$$y(t) = H[x(t)] = \alpha x(t), \quad \alpha > 0.$$

The constant α is called the **gain** of the amplifier. If $0 < \alpha < 1$, the system works as an attenuator, i.e., the amplitude of the output signal will be smaller than the amplitude of the input signal. If $\alpha > 1$, the system will amplify the input signal, i.e., the amplitude of the

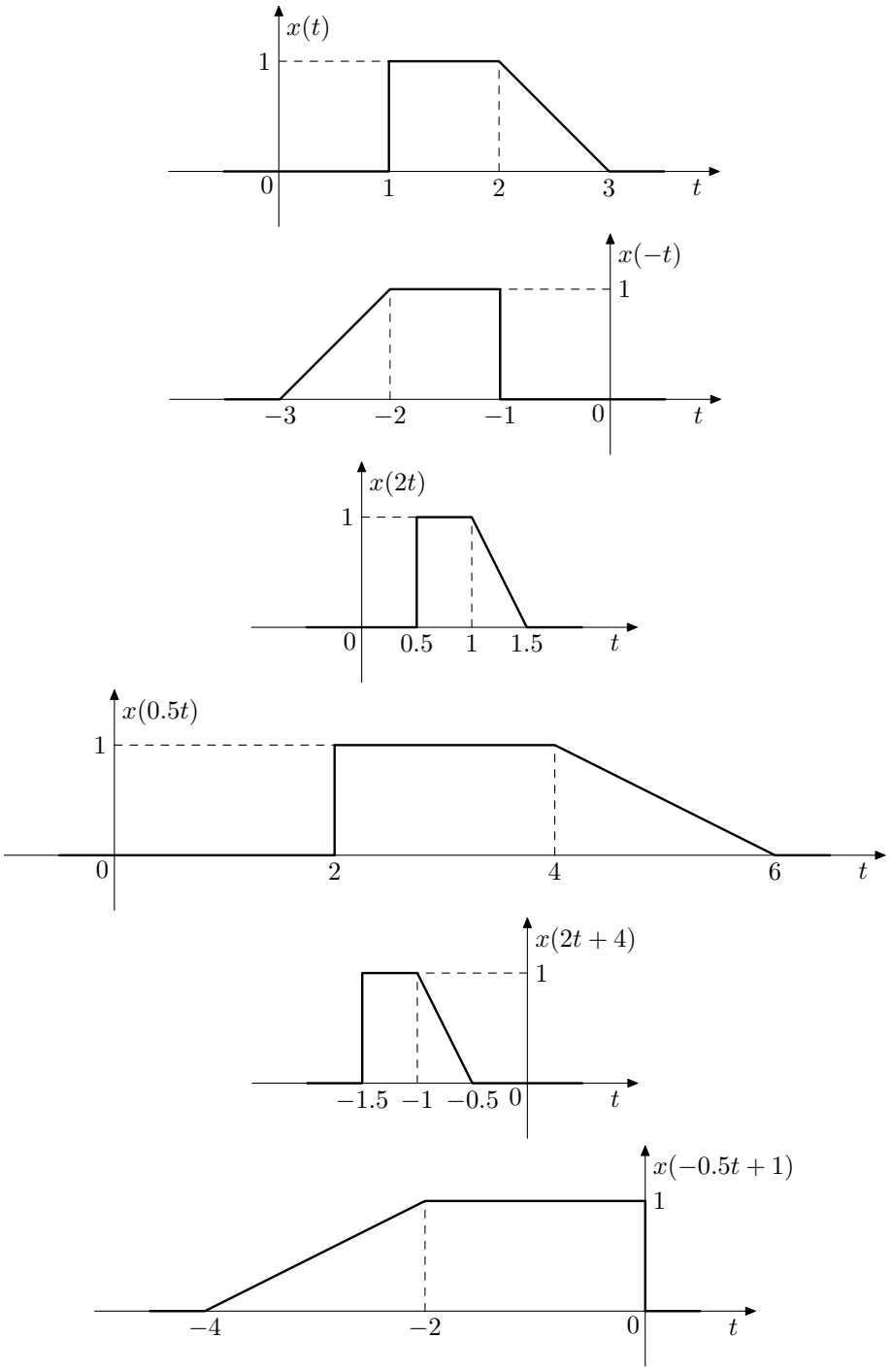


Figure 6.14: Example of the various operations on the independent variable.

output signal will be greater than the amplitude of the input signal. Fig. 6.16 shows the relation between the input signal, $x(t)$, and the output signal, $y(t)$, of an amplifier with gain α.

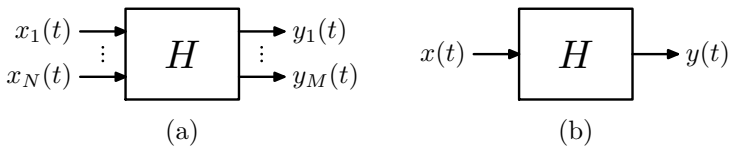


Figure 6.15: System: (a) with N inputs and M outputs; (b) with an input and an output.

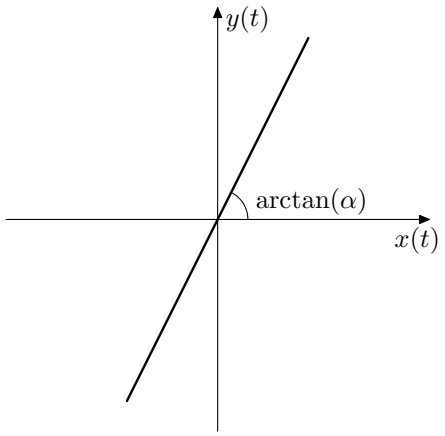


Figure 6.16: Relation between the input and output of an ideal amplifier with gain α.

The **ideal half-wave rectifier** is also an easy system to represent mathematically,

$$y(t) = H[x(t)] = \begin{cases} x(t), & x(t) \geq 0 \\ 0, & x(t) < 0 \end{cases}, \tag{6.28}$$

as well as the **ideal full-wave rectifier**,

$$y(t) = H[x(t)] = |x(t)|. \tag{6.29}$$

Fig. 6.17 shows the relation between the input and output of these two systems. Fig. 6.18 shows an example of half-wave and full-wave rectification.

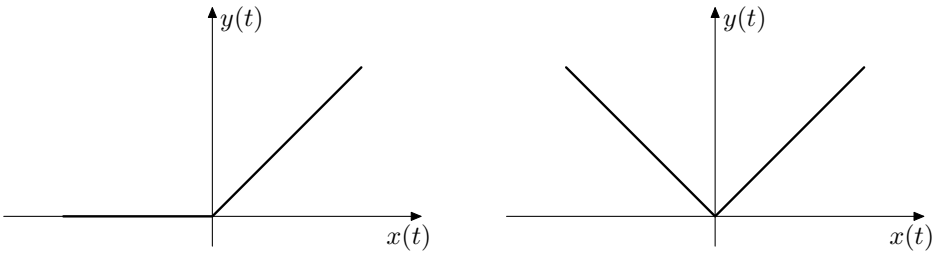


Figure 6.17: Relation between input and output of: (a) an ideal rectifier half-wave; (b) an ideal full-wave rectifier.

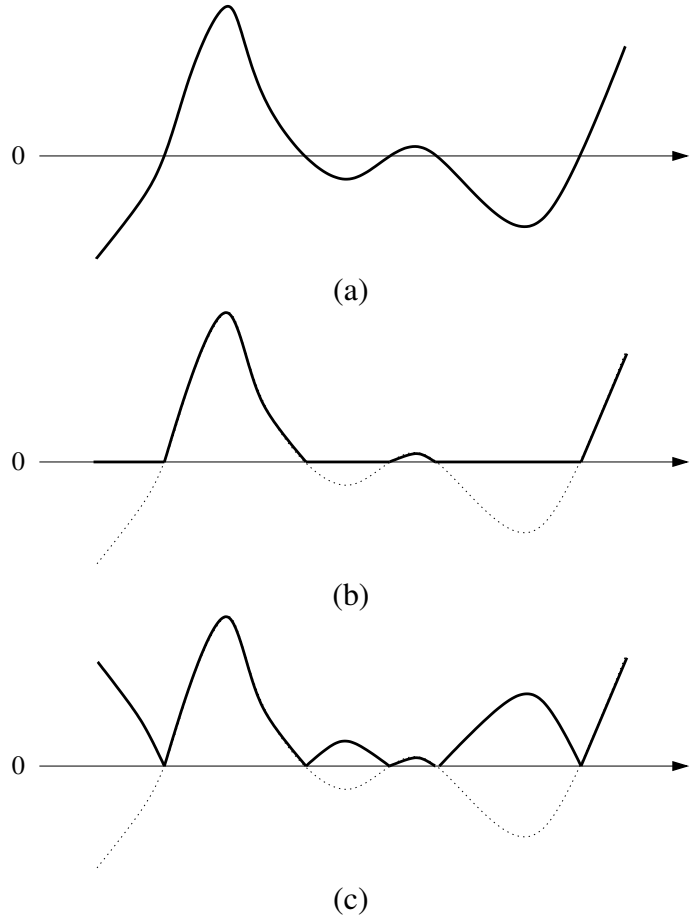


Figure 6.18: Example of rectification of a signal: (a) original signal; (b) half-wave rectification; (c) full-wave rectification.

Another example of a system that is simple to describe mathematically, although of great importance, is the **variable gain amplifier**,

$$y(t) = H[x(t)] = \alpha(t)x(t). \tag{6.30}$$

6.2.1 Properties of systems: Stability

A system is stable if, for any amplitude-limited input signal, it produces an amplitude-limited output signal, that is, if

$$|y(t)| \leq M_y < \infty, \forall x(t) : |x(t)| \leq M_x < \infty.$$

For example, the ideal amplifier is a stable system, because if

$$|x(t)| \leq M_x < \infty,$$

then

$$|y(t)| = |\alpha x(t)| = |\alpha||x(t)| \leq |\alpha| M_x < \infty, \quad \text{for} \quad |\alpha| < \infty.$$

On the other hand, the system given by

$$y(n) = 2y(n-1) + x(n)$$

is not stable. In fact, if we consider an impulse as the input of the system, i.e., making $x(n) = \delta(n)$, we will have the following sequence of output values (we assume $y(n) = 0, n < 0$):

$$\begin{aligned} &y(0) = 2y(-1) + \delta(0) = 0 + 1 = 1 \\ &y(1) = 2y(0) + \delta(1) = 2 + 0 = 2 \\ &y(2) = 2y(1) + \delta(2) = 4 + 0 = 4 \\ &\ldots \\ &y(k) = 2^k \end{aligned}$$

So, for an amplitude-limited input, we get a response that grows without limit, so the system is unstable.

6.2.2 Properties of systems: Memory

A system has memory when, to determine its output at a given instant t (or n), other input values than $x(t)$ (or $x(n)$) are needed. For example, the ideal amplifier has no memory, since to obtain the output value at time t, i.e., $y(t)$, it is only necessary to know the input value at time t, i.e., $x(t)$.

On the other hand, the system defined by the relation $y(n) = 2y(n-1) + x(n)$ has memory. In fact, to determine the value of $y(n)$, in addition to the value of $x(n)$, the output value at the previous instant is also necessary, i.e., $y(n-1)$.

The analog system defined through the relation

$$y(t) = \frac{1}{C} \int_{-\infty}^{t} x(\tau)\, d\tau \tag{6.31}$$

is also a system with memory, since its output at a given moment depends on the knowledge of the entire input signal until that moment. This system describes the behavior of the voltage, $y(t)$, at the terminals of a capacitor with capacity C, when it is supplied with a current $x(t)$.

6.2.3 Properties of systems: Causality

A system is causal if, to calculate its output at a given instant t (or n), only current or past values are needed. Therefore, a causal system does not need to know the future to determine the output at a given moment, which, physically, seems quite obvious. However, under certain

conditions, we can actually have systems that are not causal. Let us start by considering the causal system defined through

$$y(n) = x(n) + x(n-1) - 0.5y(n-1). \tag{6.32}$$

As we can see, the output of this system only depends on values from the present, $x(n)$, and from the past, $x(n-1)$ and $y(n-1)$.

Consider now the system

$$y(n) = x(n) + x(n+1) - 0.5y(n-1). \tag{6.33}$$

In this case, in addition to present and past values, a future value is also required, i.e., $x(n+1)$. However, we can transform this non-causal system into a causal system by changing the variable using $m = n + 1$,

$$y(m-1) = x(m-1) + x(m) - 0.5y(m-2). \tag{6.34}$$

Note that, in this case, the transformation of the non-causal system into a causal system was made at the cost of introducing a delay of one time unit into the system. In general, this is only possible if there are no real-time requirements, that is, if it is not necessary to have the output of the system in the shortest possible time (or within a pre-defined maximum limit) after the input was provided.

6.2.4 Properties of systems: Invertibility

A system is invertible if it is possible to determine the input signal from the output signal. Therefore, for this to be possible, each distinct input signal must produce a distinct output signal. Hence, if the system H, such that $y(t) = H[x(t)]$, is invertible, there will be an inverse system, H^{-1}, such that

$$x(t) = H^{-1}[y(t)], \quad \forall x(t). \tag{6.35}$$

For example, the system

$$y(t) = H[x(t)] = x^3(t) \tag{6.36}$$

is invertible, but the system

$$y(t) = H[x(t)] = x^2(t) \tag{6.37}$$

is not (why?).

6.2.5 Properties of systems: Linearity

Linearity is one of the two most important properties of a system, because, when verified, it allows the use of a wide range of tools for system analysis and synthesis. The other important property is the temporal invariance, which is discussed in the next section. To be linear, a system has to be **additive** and **homogeneous**.

A system is additive if

$$H[x_1(t) + x_2(t)] = H[x_1(t)] + H[x_2(t)], \quad \forall x_1(t), x_2(t). \tag{6.38}$$

This property is represented in terms of a block diagram in Fig. 6.19.

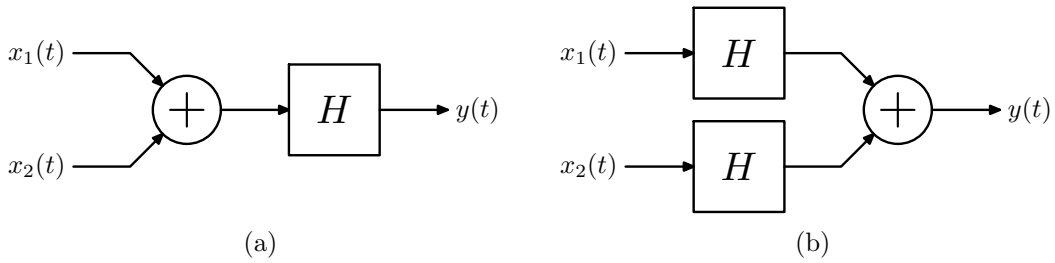


Figure 6.19: The additivity property allows permuting the sum block with the system, maintaining the same functionality.

A system is homogeneous if

$$H[\alpha x(t)] = \alpha H[x(t)], \quad \forall x(t), \alpha. \tag{6.39}$$

Fig. 6.20 illustrates this property, showing that in this case it is possible to permute the system with the gain block.

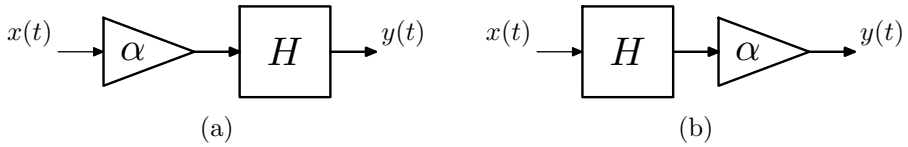


Figure 6.20: The homogeneity property allows permuting the gain block with the system, while keeping the same functionality.

6.2.6 Properties of systems: Time invariance

A system is time-invariant if a time shift of the input signal (lag or lead) produces only an equal time shift of the output signal. Therefore, a system $y(t) = H[x(t)]$ is time-invariant if

$$y(t - \tau) = H[x(t - \tau)], \quad \forall x(t), \tau. \tag{6.40}$$

As can be seen in Fig. 6.21, this means that one can switch a delay block with the system, obtaining an equivalent result.

For example, the ideal full-wave rectifier is a time-invariant system, because

$$H[x(t - \tau)] = |x(t - \tau)| = y(t - \tau). \tag{6.41}$$

On the other hand, the variable-gain amplifier is not time-invariant, because

$$H[x(t - \tau)] = \alpha(t)x(t - \tau) \neq y(t - \tau) = \alpha(t - \tau)x(t - \tau). \tag{6.42}$$

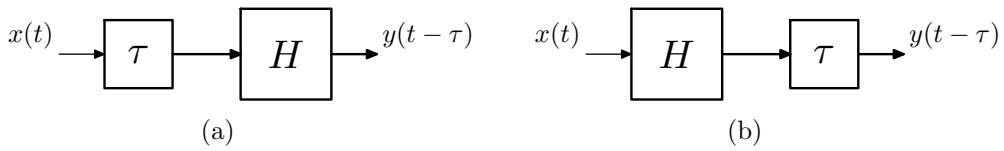


Figure 6.21: The time invariance property allows to permute the block of delay with the system.

6.3 Linear and time-invariant systems

6.3.1 Signal representation by superposition of impulses

As can be seen in Fig. 6.22, multiplying a unit impulse at $n = k$ by any discrete-time signal $x(n)$ results in a new impulse also at $n = k$ and with amplitude equal to $x(k)$, i.e.,

$$x(n)\delta(n-k) = x(k)\delta(n-k), \quad \forall k \in \mathbb{Z}.$$

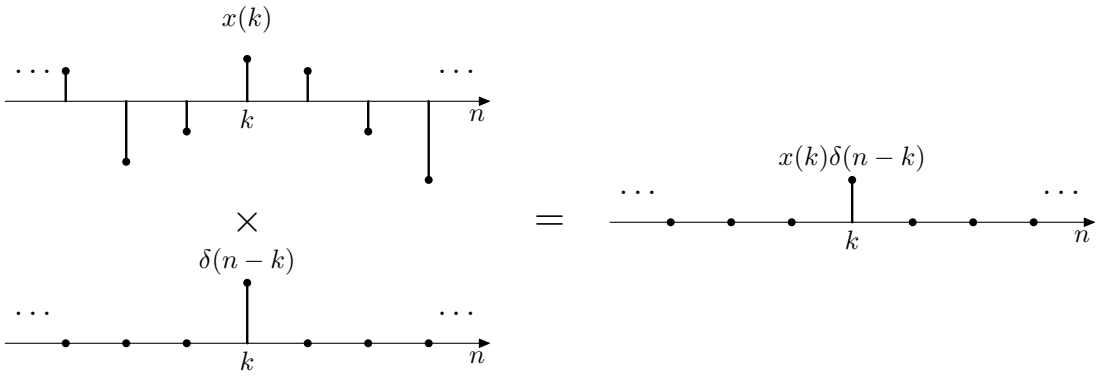


Figure 6.22: Result of multiplying a unit impulse at $n = k$ by a discrete signal $x(n)$.

Based on this observation, it is possible to represent any discrete-time signal, $x(n)$, through a sum of impulses, as exemplified in Fig. 6.23,

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k).$$

6.3.2 Impulse response

Let us now consider a linear, time-invariant discrete-time system, H, and its response to the impulse $\delta(n)$, which we will represent by $h(n)$ and call **impulse response** of the system (see Fig. 6.24). The response of system H to an arbitrary input signal, $x(n)$, can be obtained by

$$y(n) = H[x(n)] = H\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right]. \tag{6.43}$$

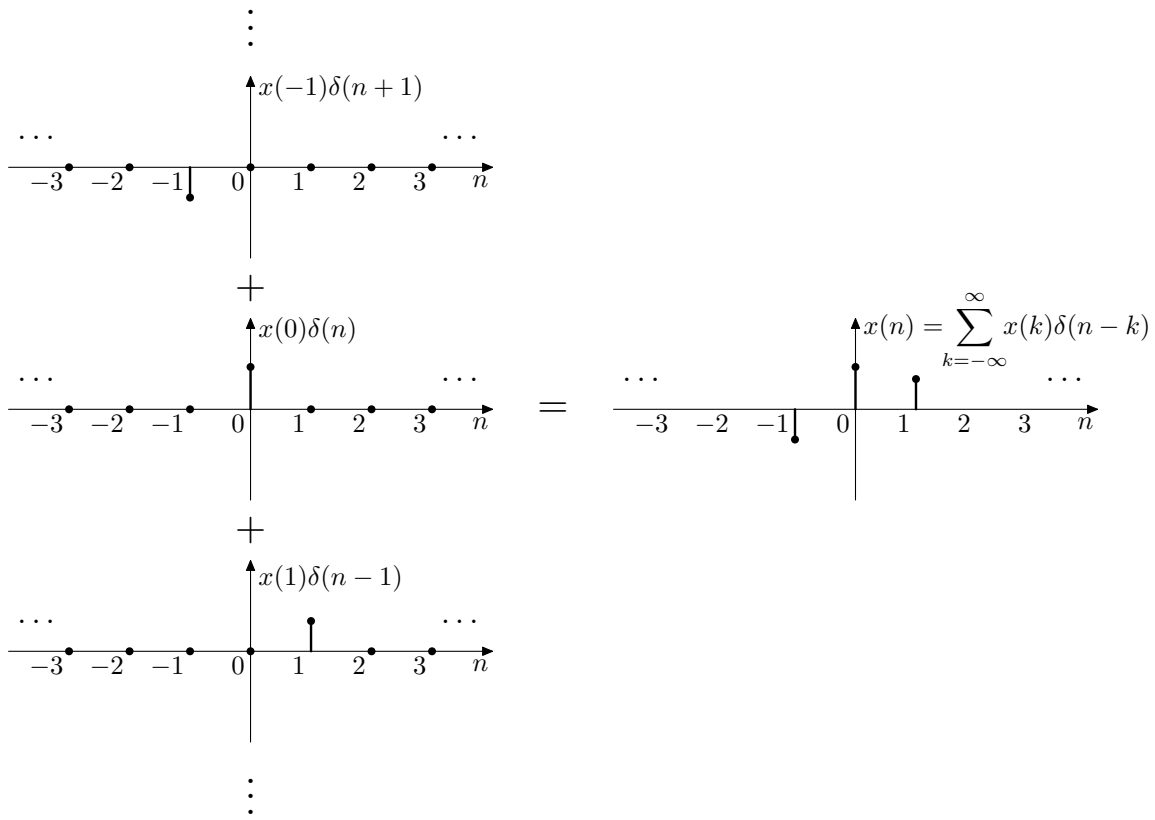


Figure 6.23: Representation of a discrete signal, $x(n)$, through a sum of impulses.

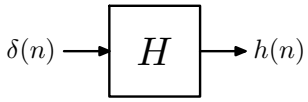


Figure 6.24: Impulse response of a system H.

As the system is linear and therefore additive, we can write

$$H\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right] = \sum_{k=-\infty}^{\infty} H[x(k)\delta(n-k)], \tag{6.44}$$

and also, using the property of homogeneity,

$$\sum_{k=-\infty}^{\infty} H[x(k)\delta(n-k)] = \sum_{k=-\infty}^{\infty} x(k)H[\delta(n-k)], \tag{6.45}$$

and finally, making use of the time-invariance property, we have

$$\sum_{k=-\infty}^{\infty} x(k)H[\delta(n-k)] = \sum_{k=-\infty}^{\infty} x(k)h(n-k). \tag{6.46}$$

This last expression shows us a way to calculate the response of any linear and time-invariant discrete-time system, with impulse response $h(n)$, to any input signal, $x(n)$. This operation is called **discrete linear convolution**, and is represented by the "$*$" operator,

$$x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k). \tag{6.47}$$

Example 6.3

Consider a system with impulse response $h(n) = 3\delta(n) + 2\delta(n-1) + \delta(n-2)$. We want to determine the response of this system to the signal $x(n) = \delta(n) - \delta(n-1)$.

The response can be calculated through

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = \tag{6.48a}$$

$$= x(0)h(n-0) + x(1)h(n-1) = h(n) - h(n-1) = \tag{6.48b}$$

$$= 3\delta(n) + 2\delta(n-1) + \delta(n-2) - 3\delta(n-1) - 2\delta(n-2) - \delta(n-3) = \tag{6.48c}$$

$$3\delta(n) - \delta(n-1) - \delta(n-2) - \delta(n-3). \tag{6.48d}$$

6.3.3 Properties of the convolution operation

We saw above that the convolution operation plays a very important role in the context of linear and time-invariant systems. Let us now look at some of its properties and their implications.

One of the most obvious properties is **commutativity**, that is,

$$a(n) * b(n) = \sum_{k=-\infty}^{\infty} a(k)b(n-k) = \sum_{k=-\infty}^{\infty} a(n-k)b(k) = b(n) * a(n), \tag{6.49}$$

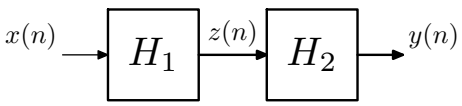
relation that can be verified through a simple change of variable.

The convolution operation is also **associative**, that is,

$$a(n) * b(n) * c(n) = a(n) * \big(b(n) * c(n)\big) = \big(a(n) * b(n)\big) * c(n). \tag{6.50}$$

Example 6.4

*We intend to determine the system equivalent to the following **series (or cascade) association** of the systems H_1 and H_2,*



Suppose that you put at the input of this cascade a delta signal, $\delta(n)$. Determine the output signal and then get the equivalent impulse response.

The convolution operation is also **distributive with respect to addition**, that is,

$$a(n) * \big(b(n) + c(n)\big) = a(n) * b(n) + a(n) * c(n). \tag{6.51}$$

(Note: use this property to determine the equivalent system in a **parallel association** of systems).

6.3.4 The impulse response and some properties of systems

Memory: A system has no memory if it only uses the actual input value to calculate the output. Therefore, since the output of a linear and time-invariant system can be calculated using

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k), \tag{6.52}$$

in a memoryless system we have $y(n) = cx(n)$, where c is an arbitrary constant, that is, $h(n) = c\delta(n)$.

Causality: A system is causal if, to calculate its output at a given instant n, only current or past input values are needed. Therefore, the output of a causal system at the instant n can only be calculated based on $x(n-k), k \geq 0$. This implies $h(n) = 0, n < 0$, i.e., the output of a linear and time-invariant causal system can be calculated through

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k). \tag{6.53}$$

Stability: As we have seen, a system is stable if it responds to amplitude-limited inputs with amplitude-limited outputs. Therefore, a linear, time-invariant discrete-time system is stable if,

$$\forall x(n) \quad \text{and} \quad |x(n)| \leq M_x < \infty,$$

we have

$$|y(n)| \leq M_y < \infty.$$

Then,

$$|y(n)| = \left| \sum_{k=-\infty}^{\infty} h(k)x(n-k) \right| \leq \sum_{k=-\infty}^{\infty} |h(k)x(n-k)| = \tag{6.54a}$$

$$\sum_{k=-\infty}^{\infty} |h(k)||x(n-k)| \leq M_x \sum_{k=-\infty}^{\infty} |h(k)|. \tag{6.54b}$$

Therefore, if the system verifies the condition

$$\sum_{k=-\infty}^{\infty} |h(k)| < \infty, \tag{6.55}$$

then it is stable.

Example 6.5
Consider a linear and time-invariant system whose impulse response is given by

$$h(n) = a^n u(n),$$

where $a \in \mathbb{R}$. We want to determine for which values of the parameter a this system is stable. As we have seen, the impulse response has to obey

$$\sum_{k=-\infty}^{\infty} |h(k)| < \infty,$$

whereby

$$\sum_{k=-\infty}^{\infty} |a^k u(k)| = \sum_{k=0}^{\infty} |a^k| < \infty,$$

i.e., $|a| < 1$.

Invertibility: A system is invertible if its input can be obtained from its output. Let us call the impulse response of the inverse system (when such a system exists) $h^{-1}(n)$. As we have seen, the response of a linear, time-invariant system can be calculated using the convolution operation. For example, in the case of discrete-time we have $y(n) = x(n) * h(n)$. The recovery of $x(n)$ from $y(n)$ is called **deconvolution**. Therefore, $h^{-1}(n)$ must be such that $x(n) = y(n) * h^{-1}(n)$, that is,

$$x(n) = y(n) * h^{-1}(n) = [x(n) * h(n)] * h^{-1}(n) = x(n) * [h(n) * h^{-1}(n)], \tag{6.56}$$

implying

$$h(n) * h^{-1}(n) = \delta(n). \tag{6.57}$$

Example 6.6
Let us consider a linear, time-invariant, discrete-time system that reproduces the input signal added to an input replica delayed by one time unit (echo), i.e.,

$$y(n) = x(n) + ax(n-1).$$

We want to determine (in case it exists) the inverse system that will eliminate the echo in the receiver.

We start by writing the impulse response of this system, which is given by

$$h(n) = \delta(n) + a\delta(n-1).$$

Then,

$$h(n) * h^{-1}(n) = [\delta(n) + a\delta(n-1)] * h^{-1}(n) = h^{-1}(n) + ah^{-1}(n-1) = \delta(n).$$

Considering that the inverse system is causal, we have $h^{-1}(n) = 0, n < 0$*. So, for* $n = 0$*, we can write*

$$h^{-1}(0) + ah^{-1}(-1) = \delta(0),$$

i.e., $h^{-1}(0) = 1$*. For* $n = 1$ *we have*

$$h^{-1}(1) + ah^{-1}(0) = \delta(1),$$

i.e., $h^{-1}(1) = -a$*. For* $n = k \geq 0$ *we have*

$$h^{-1}(k) = (-a)^k.$$

Therefore, the impulse response of the inverse system is $h^{-1}(n) = (-a)^n u(n)$*.*

Note that the inverse system is only stable if

$$\sum_{n=-\infty}^{\infty} |h^{-1}(n)| = \sum_{n=0}^{\infty} |(-a)^n| < \infty,$$

implying, in this case, that $|a| < 1$*.*

6.3.5 Response to complex exponentials

Let us consider a linear, time-invariant, discrete-time system with impulse response $h(n)$. The response of this system to the signal $x(n) = e^{\mathrm{j}\Omega n}$ is given by

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)e^{\mathrm{j}\Omega(n-k)} = e^{\mathrm{j}\Omega n} \sum_{k=-\infty}^{\infty} h(k)e^{-\mathrm{j}\Omega k} = H(e^{\mathrm{j}\Omega})e^{\mathrm{j}\Omega n}, \quad (6.58)$$

with

$$H(e^{\mathrm{j}\Omega}) = \sum_{k=-\infty}^{\infty} h(k)e^{-\mathrm{j}\Omega k}, \quad (6.59)$$

that we call **frequency response** of the system.

This result shows a very important characteristic of the linear and time-invariant systems, which will serve as the basis for other results that we will present later: the response to the signal $e^{\mathrm{j}\Omega n}$ is the signal $H(e^{\mathrm{j}\Omega})e^{\mathrm{j}\Omega n}$, where $H(e^{\mathrm{j}\Omega}) \in \mathbb{C}$ is given by (6.59). For this reason, we say that complex exponentials are **eigensignals** of the linear and time-invariant systems.

As we have seen, $H(e^{\mathrm{j}\Omega})$ depends, on the one hand, on the system itself and, on the other hand, on the frequency, Ω, of the complex exponential at the input of the system. Therefore, for a given system, the value of $H(e^{\mathrm{j}\Omega})$ only depends on the frequency of the signal.

Example 6.7

Consider a system with impulse response

$$h(n) = \begin{cases} 0.5, & n = 0, 1 \\ 0, & \textit{others} \end{cases}.$$

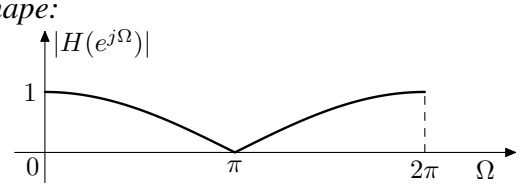
The frequency response of this system is given by

$$H(e^{\mathrm{j}\Omega}) = \sum_{n=-\infty}^{\infty} h(n)e^{-\mathrm{j}\Omega n} = \frac{1}{2} + \frac{1}{2}e^{-\mathrm{j}\Omega} = \frac{1}{2}e^{-\mathrm{j}\Omega/2}(e^{\mathrm{j}\Omega/2} + e^{-\mathrm{j}\Omega/2}) = e^{-\mathrm{j}\Omega/2}\cos\frac{\Omega}{2}.$$

Since, in general, $H(e^{\mathrm{j}\Omega}) \in \mathbb{C}$, it is usual to calculate its modulus and phase. In this case, the modulus is given by

$$|H(e^{\mathrm{j}\Omega})| = \left|e^{-\mathrm{j}\Omega/2}\cos\frac{\Omega}{2}\right| = \left|\cos\frac{\Omega}{2}\right|,$$

which has the following shape:



Example 6.8

Consider now a system with impulse response

$$h(n) = \begin{cases} 0.5, & n = 0 \\ -0.5, & n = 1 \\ 0, & \textit{others} \end{cases}.$$

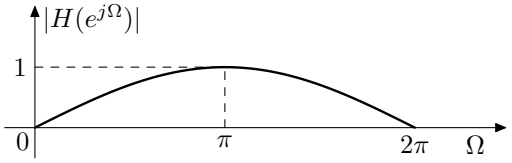
The frequency response of this system is given by

$$H(e^{\mathrm{j}\Omega}) = \sum_{n=-\infty}^{\infty} h(n)e^{-\mathrm{j}\Omega n} = \frac{1}{2} - \frac{1}{2}e^{-\mathrm{j}\Omega} = \frac{1}{2}e^{-\mathrm{j}\Omega/2}(e^{\mathrm{j}\Omega/2} - e^{-\mathrm{j}\Omega/2}) = \mathrm{j}e^{-\mathrm{j}\Omega/2}\sin\frac{\Omega}{2}.$$

Its modulus is

$$|H(e^{\mathrm{j}\Omega})| = \left|\mathrm{j}e^{-\mathrm{j}\Omega/2}\sin\frac{\Omega}{2}\right| = \left|\sin\frac{\Omega}{2}\right|,$$

which has the following shape:



6.4 The Z transform

Let us consider a discrete system, H, linear and time-invariant, with impulse response $h(n)$, and the signal

$$x(n) = z^n = \left(re^{\mathrm{j}\Omega}\right)^n.$$

The response of H to $x(n)$ can be obtained through the convolution sum, i.e.,

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) = \sum_{k=-\infty}^{\infty} h(k)z^{n-k} = z^n \sum_{k=-\infty}^{\infty} h(k)z^{-k} = z^n H(z).$$

The function

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n} \tag{6.60}$$

is the Z **transform** of $h(n)$.

Example 6.9

Calculate the Z transform, $X(z)$, of the signal $x(n) = a^n u(n)$.

We have

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} = \sum_{n=0}^{\infty}(az^{-1})^n = \frac{1}{1-az^{-1}}, \quad |z| > |a|.$$

Note that a convergence region is associated with $X(z)$, that is, in this case the function $X(z)$ obtained is only valid for values of z such that $|z| > |a|$.

Example 6.10

Calculate the Z transform of the signal $x(n) = a^n \cos(\Omega n)u(n)$.

In this case $X(z)$ is given by

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} = \frac{1}{2}\sum_{n=0}^{\infty}(ae^{\mathrm{j}\Omega}z^{-1})^n + (ae^{-\mathrm{j}\Omega}z^{-1})^n = \tag{6.61a}$$

$$\frac{1}{2}\left(\frac{1}{1-ae^{\mathrm{j}\Omega}z^{-1}} + \frac{1}{1-ae^{-\mathrm{j}\Omega}z^{-1}}\right), \quad |z| > |a|, \tag{6.61b}$$

which can be simplified to

$$X(z) = \frac{1 - a\cos\Omega z^{-1}}{1 - 2a\cos\Omega z^{-1} + a^2 z^{-2}}. \tag{6.62}$$

6.4.1 The time shift property

Let us consider a signal, $x(n)$, with Z transform $X(z)$. We want to determine the Z transform of the signal $x(n-k)$, that is,

$$\sum_{n=-\infty}^{\infty} x(n-k)z^{-n} = \sum_{n=-\infty}^{\infty} x(n)z^{-n-k} = z^{-k}\sum_{n=-\infty}^{\infty} x(n)z^{-n} = z^{-k}X(z).$$

Therefore, if

$$x(n) \overset{\mathcal{Z}}{\longleftrightarrow} X(z),$$

then

$$x(n-k) \overset{\mathcal{Z}}{\longleftrightarrow} z^{-k}X(z).$$

This property allows an easy transformation to the Z domain of linear difference equations of constant coefficients, that is, of the type

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{k=0}^{M} b_k x(n-k),$$

since, because

$$y(n-k) \overset{\mathcal{Z}}{\longleftrightarrow} z^{-k}Y(z) \quad \text{and} \quad x(n-k) \overset{\mathcal{Z}}{\longleftrightarrow} z^{-k}X(z),$$

we have

$$\sum_{k=0}^{N} a_k y(n-k) \overset{\mathcal{Z}}{\longleftrightarrow} \sum_{k=0}^{N} a_k z^{-k}Y(z) \quad \text{and} \quad \sum_{k=0}^{M} b_k x(n-k) \overset{\mathcal{Z}}{\longleftrightarrow} \sum_{k=0}^{M} b_k z^{-k}X(z)$$

and

$$\sum_{k=0}^{N} a_k z^{-k}Y(z) = \sum_{k=0}^{M} b_k z^{-k}X(z).$$

We can also write

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\displaystyle\sum_{k=0}^{M} b_k z^{-k}}{\displaystyle\sum_{k=0}^{N} a_k z^{-k}}.$$

If $x(n)$ is the input signal and $y(n)$ is the respective output signal of a certain linear, time-invariant system, then the function $H(z)$ is the **transfer function** of that system.

Example 6.11
Determine the transfer function of a system described by the equation

$$y(n) = x(n) - x(n-1) + 0.5y(n-1).$$

Calculating the Z transform term by term, we obtain

$$Y(z) = X(z) - z^{-1}X(z) + 0.5z^{-1}Y(z)$$

and, hence,

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-1}}{1 - 0.5z^{-1}}.$$

The roots of the numerator and denominator of $H(z)$ are called, respectively, **zeros** and **poles** of the transfer function. A system that does not have poles (i.e., the denominator of $H(z)$ is a constant) is called a **FIR** system (finite impulse response system), otherwise it is called an **IIR** system (infinite impulse response system).

Example 6.12
Determine the poles and zeros of the transfer function of the system shown in Example 6.11.

The poles of a system are the roots of the denominator of the transfer function, so, in this case, we have

$$1 - 0.5z^{-1} = 0 \Leftrightarrow z = 0.5.$$

The zeros of a system are the roots of the numerator of the transfer function, so, in this case, we have

$$1 - z^{-1} = 0 \Leftrightarrow z = 1.$$

Therefore, the system has a pole at $z = 0.5$ and a zero at $z = 1$.

The transfer function, $H(z)$, of a certain system, corresponds to the Z transform of the impulse response, $h(n)$, of that same system, that is,

$$h(n) \overset{\mathcal{Z}}{\longleftrightarrow} H(z). \tag{6.63}$$

In fact, because

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k),$$

then

$$Y(z) = \sum_{n=-\infty}^{\infty} y(n)z^{-n} = \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h(k)x(n-k)z^{-n} = \tag{6.64a}$$

$$\sum_{k=-\infty}^{\infty} h(k) \sum_{n=-\infty}^{\infty} x(n-k)z^{-n} = \sum_{k=-\infty}^{\infty} h(k) \sum_{n=-\infty}^{\infty} x(n)z^{-(n+k)} = \tag{6.64b}$$

$$X(z) \sum_{k=-\infty}^{\infty} h(k)z^{-k} = X(z)H(z). \tag{6.64c}$$

This result shows the relationship between the multiplication operation in one domain and the convolution operation in the other domain.

6.4.2 Inversion of the Z transform

The inversion of generic Z transforms involves the computation of integrals in the complex plane, a topic that goes beyond what we intend to cover in this text. However, there are particular cases whose inversion is easy to determine by alternative methods. Next, we show one of these cases.

As we saw above, systems described by linear difference equations of constant coefficients lead to rational transfer functions, constituted by quotients between polynomials in z. These systems are very important, as they allow us to describe many practical cases of interest. In this case we have

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{b_0 \prod_{k=1}^{M} (1 - z_k z^{-1})}{a_0 \prod_{k=1}^{N} (1 - p_k z^{-1})},$$

where the z_k represent the M zeros and the p_k the N poles of the system, and $z_k, p_k \neq 0, \forall k$. If $N > M$ and if there are no multiple poles, that is, if $p_k \neq p_l, \forall k \neq l$, we can write

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - p_k z^{-1}},$$

which corresponds to the expansion of $H(z)$ in partial fractions. If $N \leq M$, we can still use this method, but after reducing the degree of the numerator by polynomial division. Note that the need for $H(z)$ not to have zeros or poles in $z = 0$ results from the need for $a_0, b_0 \neq 0$. Obviously, this restriction can easily be overcome by factoring these terms.

Note that, because (see Example 6.9)

$$\frac{A_k}{1 - p_k z^{-1}} \overset{\mathcal{Z}}{\longleftrightarrow} A_k p_k^n u(n),$$

obtaining the inversion is now trivial.

Example 6.13
We intend to invert the Z transform

$$H(z) = \frac{1 - 1.7z^{-1}}{1 - 2.05z^{-1} + z^{-2}}.$$

Because

$$\frac{1 - 1.7z^{-1}}{1 - 2.05z^{-1} + z^{-2}} = \frac{1 - 1.7z^{-1}}{(1 - 1.25z^{-1})(1 - 0.8z^{-1})} = \frac{A_1}{1 - 1.25z^{-1}} + \frac{A_2}{1 - 0.8z^{-1}},$$

we obtain

$$h(n) = A_1\ 1.25^n u(n) + A_2\ 0.8^n u(n).$$

The A_1 and A_2 constants can be obtained using

$$A_1 = \left.\frac{1 - 1.7z^{-1}}{1 - 0.8z^{-1}}\right|_{z=1.25} = -1$$

and

$$A_2 = \left.\frac{1 - 1.7z^{-1}}{1 - 1.25z^{-1}}\right|_{z=0.8} = 2.$$

Example 6.14
Calculate the impulse response of the system

$$H(z) = \frac{1}{z^2 - 1}.$$

Since the impulse response of a discrete-time system can be obtained through the inverse Z transform of the transfer function of that system, then

$$H(z) = \frac{1}{z^2 - 1} = \frac{z^{-2}}{1 - z^{-2}} = z^{-2}G(z),$$

and

$$G(z) = \frac{1}{1 - z^{-2}} = \frac{1}{(1 - z^{-1})(1 + z^{-1})} = \frac{1}{2}\left(\frac{1}{1 - z^{-1}} + \frac{1}{1 + z^{-1}}\right),$$

then

$$g(n) = \frac{1}{2}\big(1^n + (-1)^n\big)u(n)$$

and

$$h(n) = \frac{1}{2}\big(1 + (-1)^{n-2}\big)u(n-2).$$

Example 6.15
Calculate the response of the system with transfer function

$$H(z) = \frac{1 - 0.2z^{-1}}{1 + 0.5z^{-1}}$$

to the signal $x(n) = 0.2^n u(n)$.

Since $Y(z) = H(z)X(z)$*, and because*

$$x(n) = 0.2^n u(n) \overset{z}{\longleftrightarrow} \frac{1}{1 - 0.2z^{-1}},$$

then

$$Y(z) = \frac{1 - 0.2z^{-1}}{1 + 0.5z^{-1}} \frac{1}{1 - 0.2z^{-1}} = \frac{1}{1 + 0.5z^{-1}}.$$

Therefore, the response of the system is

$$y(n) = (-0.5)^n u(n).$$

6.4.3 Stability

As we have seen, under certain conditions, the transfer function of a system can be described by

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - p_k z^{-1}},$$

where the p_k represent the (simple) poles of the system. Since the impulse response of the system, $h(n)$, corresponds to the inverse Z transform of $H(z)$, then

$$h(n) = \sum_{k=1}^{N} A_k p_k^n u(n).$$

In general, $h(n)$ only converges if all the terms of the sum converge, for which it is necessary that $|p_k| < 1, \forall k$. In fact, as we saw earlier when we discussed the properties of the impulse response, for a system to be stable we must have

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty,$$

so for this condition to be verified we have to ensure that $|p_k| < 1, \forall k$, that is, all poles of the system must have a modulus lower than one.

Example 6.16
Determine the stability of the system presented in Example 6.13.

The poles of this system are $p_1 = 1.25$ *and* $p_2 = 0.8$*. Since* $|p_1| > 1$*, then the system is not stable. In fact, this pole contributes a term in the impulse response (*$1.25^n u(n)$*) that grows without limit when* $n \to \infty$*.*

6.4.4 Frequency response

Since, for systems described by linear difference equations of constant coefficients, we have

$$H(z) = \frac{\displaystyle\sum_{k=0}^{M} b_k z^{-k}}{\displaystyle\sum_{k=0}^{N} a_k z^{-k}} = \frac{z^{-M}\displaystyle\sum_{k=0}^{M} b_k z^{M-k}}{z^{-N}\displaystyle\sum_{k=0}^{N} a_k z^{N-k}} = z^{N-M}\frac{b_0\displaystyle\prod_{k=1}^{M}(z-z_k)}{a_0\displaystyle\prod_{k=1}^{N}(z-p_k)},$$

then,

$$|H(e^{\mathrm{j}\Omega})| = \left|\frac{b_0}{a_0}\right| \frac{\displaystyle\prod_{k=1}^{M}|e^{\mathrm{j}\Omega}-z_k|}{\displaystyle\prod_{k=1}^{N}|e^{\mathrm{j}\Omega}-p_k|}.$$

Note that this last expression can be interpreted as the quotient between the product of the distances from the point $e^{\mathrm{j}\Omega}$ to the zeros of $H(z)$ and the product of the distances from the point $e^{\mathrm{j}\Omega}$ to the poles of $H(z)$.

Example 6.17

We intend to calculate the modulus of the frequency response of the system whose transfer function is

$$H(z) = \frac{1}{1-az^{-1}}.$$

We have

$$|H(e^{\mathrm{j}\Omega})| = \frac{1}{|1-ae^{-\mathrm{j}\Omega}|}.$$

For example, the gain of this system for $f = 0$ is

$$|H(e^{0})| = |H(1)| = \frac{1}{|1-a|},$$

and for $f = f_s/2$ is

$$|H(e^{\mathrm{j}\pi})| = |H(-1)| = \frac{1}{|1+a|}.$$

For $f = f_s/4$ we have

$$|H(e^{\mathrm{j}\pi/2})| = |H(\mathrm{j})| = \frac{1}{|1+\mathrm{j}a|} = \frac{1}{\sqrt{1+a^2}}.$$

Note that, if $0 < a < 1$, the system has a low-pass behavior (see Fig. 6.25), whereas if $-1 < a < 0$, then the system has a high-pass characteristic (see Fig. 6.26).

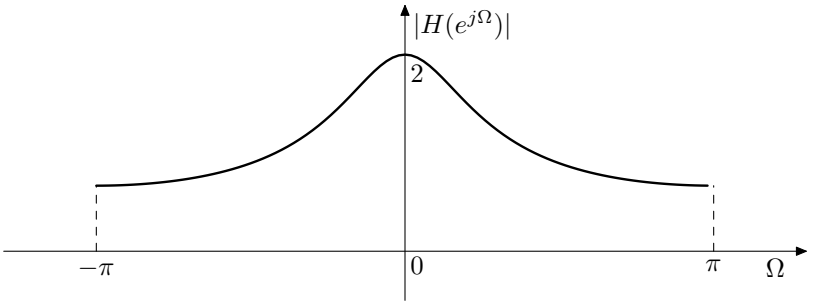


Figure 6.25: Frequency response of a digital low-pass filter with a pole at $z = 0.5$.

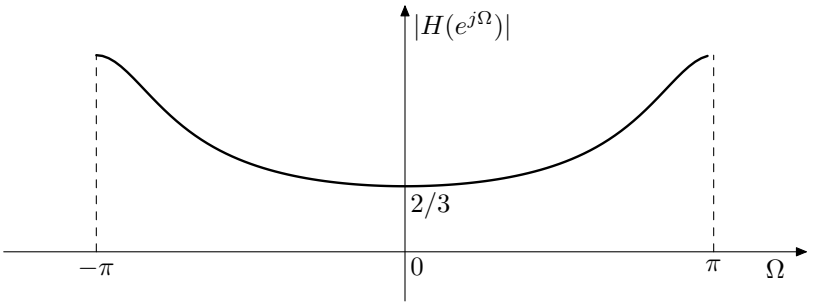


Figure 6.26: Frequency response of a digital high-pass filter with a pole at $z = -0.5$.

6.4.5 Relation between the Z transform and the discrete-time Fourier transform

Making $z = e^{\mathrm{j}\Omega}$ we get

$$X(z)|_{z=e^{\mathrm{j}\Omega}} = X(e^{\mathrm{j}\Omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-\mathrm{j}\Omega n},$$

which corresponds to the **discrete-time Fourier transform (DTFT)** of $x(n)$. However, this is only possible if the convergence region of $X(z)$ includes the unit circle (that is, all points in the complex plane with modulus equal to one).

In fact, the direct relation (of analysis) of the discrete-time Fourier transform is given by

$$X(F) = \sum_{n=-\infty}^{\infty} x(n)e^{-\mathrm{j}2\pi Fn}, \tag{6.65}$$

and the inverse relation (of synthesis) by

$$x(n) = \int_0^1 X(F)e^{\mathrm{j}2\pi Fn}dF. \tag{6.66}$$

For the case of normalized frequency in radians, we have

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-\mathrm{j}\Omega n}, \tag{6.67}$$

and

$$x(n) = \frac{1}{2\pi} \int_0^{2\pi} X(\Omega) e^{\mathrm{j}\Omega n} d\Omega. \tag{6.68}$$

Note that the function $X(F)$ is periodic, with a unit period, i.e.,

$$X(F+k) = \sum_{n=-\infty}^{\infty} x(n) e^{-\mathrm{j}2\pi(F+k)n} = \sum_{n=-\infty}^{\infty} x(n) e^{-\mathrm{j}2\pi Fn} e^{-\mathrm{j}2\pi kn} = X(F).$$

7 Predictive coding

See slides.

8 Transform coding

See slides.

9 Examples of audio coding

See slides.

10 Example of image coding

See slides.

11 Example of video coding

See slides.

References

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, **58**(2), 345–363.

Cutland, N. J. (1980). *Computability: An introduction to recursive function theory*. Cambridge University Press.

Golomb, S. (1966). Run-length encodings. *IEEE Trans. on Information Theory*, **12**(3), 399–401.

Hamming, R. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, **29**(2), 147–160.

Hartley, R. V. L. (1928). Transmission of information. *Bell System Technical Journal*, **7**(3), 535–563.

Huffman, D. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, **40**(9), 1098–1101.

Minsky, M. L. (1967). *Computation: Finite and infinite machines*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Nyquist, H. (1924). Certain factors affecting telegraph speed. *Trans. of the American Institute of Electrical Engineers*, **XLIII**, 412–422.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, **27**, 379–423,623–656.

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell Labs Technical Journal*, **30**(1), 50–64.

Sipser, M. (2012). *Introduction to the theory of computation*. Cengage Learning, 3rd edition.

Storer, J. A. and Szymanski, T. G. (1982). Data compression via textual substitution. *Journal of the ACM*, **29**(4), 928–951.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **42**(2), 230–265.

Welch, T. A. (1984). A technique for high-performance data compression. *IEEE Computer*, **17**(6), 8–19.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, **23**, 337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Trans. on Information Theory*, **24**(5), 530–536.