LECTURES IN DIGITAL COMMUNICATIONS

LECTURES IN DIGITAL COMMUNICATIONS
Information Theory Basics

Marco Luise
Università di Pisa, November 24, 2025

Una pubblicazione dell'UNIVERSITÁ DI PISA

PREFACE

”But surpassing all stupendous inventions, what sublimity of mind was his who dreamed of finding means to communicate his deepest thoughts to any other person, though distant by mighty intervals of place and time.”

— Galileo Galilei, Dialogo sopra i due massimi sistemi del mondo,

Florence 1632, trans. by Albert Van Helden

Thanks to (random order): Silvia, Umberto Mengali, Emerson Lake & Palmer, Riccardo De Gaudenzi, Ezio Biglieri, Franco Russo, Luca Sanguinetti, Giorgio Vitetta, Stefano&Roberta, Nino D'Amico, my many students.

CONTENTS

CHAPTER 1

DIGITAL IS COMMUNICATING

"Captain Kirk to Enterprise, Captain Kirk to Enterprise."

—Star Trek TV movies series, 1966-1969

Captain Kirk holds in his hands a device that back in the sixties was totally Sci-Fi, while today, only 60 years later, is the workhorse of 21st-Century digital technologies: in his parlance, the "Communicator"; in ours, a "Smartphone"

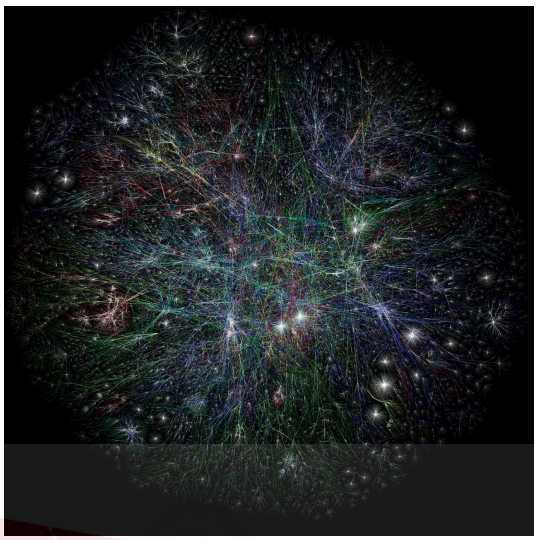


Figure 1.1 *The Black Cloud*: pictorial representation of the most relevant Internet interconnections across the world

1.1 Internet and "The Black Cloud"

Fred Hoyle (1915-2001) was a great British scientist, the inventor of the term "Big Bang" to indicate the origin of the Universe, a curiously ironic coined term to counter the idea of the "birth" of something that Hoyle himself believed to be immanent and unchanging.

Hoyle was also a great science fiction writer, and his masterpiece is undoubtedly "The Black Cloud," published in 1957, which tells the imaginary story of an enormous cloud of interstellar gas, a cloud that approaches the Earth and surrounds it, blocking the sun's rays and endangering life on the planet. A group of scientists eventually understand that the cloud is actually a living organism, and manages to communicate with this interstellar being, endowed with a highly advanced form of intelligence (and amazed that intelligent life could exist on a planet). The Black Cloud realizes the damage it is doing and abandons Earth after unsuccessfully attempting to instill a superhuman level of knowledge in the lead scientist Chris Kingsley, through ultra-fast sensory messages. Kingsley does not survive the shock.

During one of the interviews, the Black Cloud lucidly explains that its own great intelligence derives from a multitude of elementary minds that amplify their individual intellects through ramified and almost instantaneous communication links (today we would say, in Internet parlance, a mesh network) that effectively create a single mind innervated by a dense network: intelligence out of communications.

I read this book as a kid and was fascinated by it – every now and then I take the Black Cloud (I have both the original English edition and the Italian translation by Garzanti, which I think is a rarity) out of my bookshelf, I sift through it, I turn it over in my hands like an object of worship. What better than the black cloud could represent the idea we have of the Internet and of Artificial Intelligence today? The omniscience of it? After all, every time we want to represent the Internet on a sheet of paper, in a diagram, we draw a Cloud...

And what would the Internet be without the communication technologies that infuse and constitute it?

Figure 1.2 Claude E. Shannon (1916-2001)

1.2 On the giant's shoulders

It all began in the 1940s with a series of scientific contributions by Claude E. Shannon, top secret at the time and later made public – the *incipit* of it all is shown in Fig. 1.3. The emerging research field, which also included the great Norbert Wiener and John von Neumann, was at the time called "cybernetics," but Shannon soon found his own specific way, and what he invented is now recognized as the foundation of what we now call the *information age*. Whenever we receive a video clip via WhatsApp today, we know that it is of poorer quality than the original because it has been compressed to save space on our smartphone. We're also concerned that this video, perhaps of a personal nature, might be posted online by someone because our phone has been hacked. If, however, we've already saved the video on YouTube and we watch it again on our smartphone, it might freeze or become grainy because the wireless connection quality isn't optimal.

We have just listed the three everyday main concerns that we have when dealing with digital data: the three issues considered by Shannon with an absolutely visionary approach, half a century ahead of its time: *efficiency*, *security*, and *robustness* of information and digital, data. These issues are dealt with by Information Theory and, after being identified some 65 years ago, have now entered our daily lives decades later – a sign of Engineering with a capital "E."

1.3 Being Digital

Thirty years ago (!), in 1995, Nicholas Negroponte published "Being Digital", which shook the general public and confirmed what researchers in the field already knew: the future is digital, and anyone who isn't digital risks being swept away by the ebb tide.

Thirty years later, the message of Negroponte (another visionary...) is even more valid, and has fortunately been received by many segments of our society. Information Theory belongs to "immortal" engineering and "immortal" computer science – is it a set of "technicalities"? No. Not only that. Information Theory has introduced a large number of ideas

Reprinted with corrections from *The Bell System Technical Journal*,
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

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[1] and Hartley[2] 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.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

If the number of messages in the set is finite then this number or any monotonic function of this number can be regarded as a measure of the information produced when one message is chosen from the set, all choices being equally likely. As was pointed out by Hartley the most natural choice is the logarithmic function. Although this definition must be generalized considerably when we consider the influence of the statistics of the message and when we have a continuous range of messages, we will in all cases use an essentially logarithmic measure.

The logarithmic measure is more convenient for various reasons:

1. It is practically more useful. Parameters of engineering importance such as time, bandwidth, number of relays, etc., tend to vary linearly with the logarithm of the number of possibilities. For example, adding one relay to a group doubles the number of possible states of the relays. It adds 1 to the base 2 logarithm of this number. Doubling the time roughly squares the number of possible messages, or doubles the logarithm, etc.

2. It is nearer to our intuitive feeling as to the proper measure. This is closely related to (1) since we intuitively measures entities by linear comparison with common standards. One feels, for example, that two punched cards should have twice the capacity of one for information storage, and two identical channels twice the capacity of one for transmitting information.

3. It is mathematically more suitable. Many of the limiting operations are simple in terms of the logarithm but would require clumsy restatement in terms of the number of possibilities.

The choice of a logarithmic base corresponds to the choice of a unit for measuring information. If the base 2 is used the resulting units may be called binary digits, or more briefly *bits*, a word suggested by J. W. Tukey. A device with two stable positions, such as a relay or a flip-flop circuit, can store one bit of information. N such devices can store N bits, since the total number of possible states is 2^N and $\log_2 2^N = N$. If the base 10 is used the units may be called decimal digits. Since

$$\begin{aligned}\log_2 M &= \log_{10} M / \log_{10} 2 \\ &= 3.32 \log_{10} M,\end{aligned}$$

[1] Nyquist, H., "Certain Factors Affecting Telegraph Speed," *Bell System Technical Journal*, April 1924, p. 324; "Certain Topics in Telegraph Transmission Theory," *A.I.E.E. Trans.*, v. 47, April 1928, p. 617.
[2] Hartley, R. V. L., "Transmission of Information," *Bell System Technical Journal*, July 1928, p. 535.

Figure 1.3 The birth act of the Digital WOrdl and of Information Theory

and concepts, as I have already mentioned, relating to language, privacy, and efficiency, which are now common knowledge and daily life for every citizen of the modern Digital World. That's why anyone who wants to professionally deal with the digital world must have a real deep understanding of such topics.

1.4 These lecture notes...

... are the result of more than a decade of teaching Digital Communications and Information Theory at the University of Pisa. They are written in non-native, international English, and cover classical and modern topics in the field of communication and data processing technologies within the Internet, with a mix of Information Theory and Digital Communications, and a sprinkling of Signals and Systems. This mix also represents my personal experience and knowledge, which I enjoy sharing with my students. It would be unfair to write about things I don't know or I have not used in research or profession. The choice of topics and level of depth is therefore entirely subjective and obviously not exhaustive. I think, however, that they represent a good overview of the scientific and technological foundations that even today, decades after their invention, are relevant and not contingent, a perennial cultural heritage of digital civilization.

Any comments or suggestions are welcome at the email address `marco.luise@unipi.it`.
Enjoy the reading.

CHAPTER 2

1-2-3 OF SIGNALS SYSTEMS COMMUNICATIONS

"Le système Chappe: The world's first (wireless) telegraph network developed by Napoleon, carrying digitally-encoded text messages across 19h-Century France via optical signals relayed by a network of repeating stations extended from Lyon to Venice"

—Optical telegraph station next to the Rohan Castle in Saverne, France, renovated in 1998

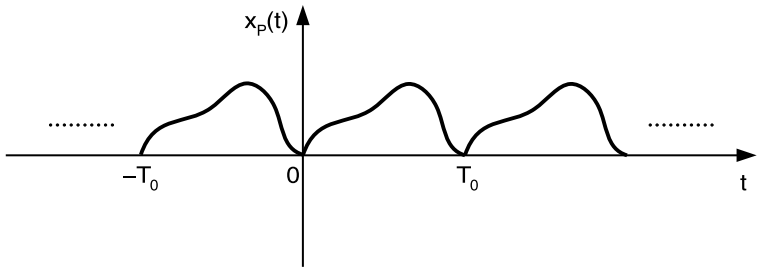


Figure 2.1 Example of a periodic signal

The motto of this chapter is: back to the basics. Its aim is in fact reviewing the main concepts related to the nature and processing of analog and digital signals, something that the experienced reader may either skip with no harm or read with renovated pleasure.

2.1 Basics of Fourier analysis of analog signals

The "Swiss Knife" of every communications engineer dealing with (wireless) systems design is *Fourier analysis*. We do not pretend to perform a comprehensive review of such a huge and fundamental topic. We just want to re-state here the main results and settle a notation concerning the analysis of time-continuous and time-discrete signals that will be the foundation of many, many concepts and tools we will extensively use in the next Chapters.

2.1.1 Periodic Signals and the Fourier Series

We've been taught back in primary schools that white light is a combination of all colors. This is a very first example of Fourier analysis. To be a little bit more specific, we know that every periodic signal $x_p(t)$ (i.e., such that $x_p(t) = x_p(t + T_0)$ for some $T_0 > 0$ that is called *repetition period* as in Fig. 2.1) can be decomposed into a sum of simpler periodic signals, namely, sinusoids as follows:

$$x_p(t) = A_0 + A_1 \cos(2\pi f_0 t + \theta_1) + A_2 \cos(2\pi 2 f_0 t + \theta_2) + ...$$
$$+A_k \cos(2\pi k f_0 t + \theta_k) + ... \quad (2.1)$$

Apart from the constant, DC value, A_0, it is seen that the sinusoids oscillate at frequencies kf_0, the so-called *harmonic frequencies*, that are integer multiples of the *fundamental* or *repetition* frequency $f_0 = 1/T_0$. The k-th component of the expansion (2.1) bears an amplitude A_k, and a phase θ_k. Whilst the value of the oscillation frequencies are always the same for any T_0-periodic signal, the specific values of A_k and θ_k do depend on the shape of the actual $x_p(t)$ under analysis. The sequence of coefficients A_k is called the *amplitude spectrum* of $x_p(t)$, and the sequence of the phases θ_k is the *phase spectrum*. Knowledge of the amplitude and phase spectra is equivalent to knowledge of the signal itself, since based on that knowledge we can from (2.1) *synthesize* back the signal in the time domain. This is why (2.1) is called the *synthesis* equation. And this is why the popular musical instrument MiniMoog of the glorious 70's was called a *synthesizer*: it simply implemented (2.1) by heroic low-cost analog hardware to generate (arbitrary) periodic waveforms to be used in musical compositions as a replacement of naturally-generated sounds.

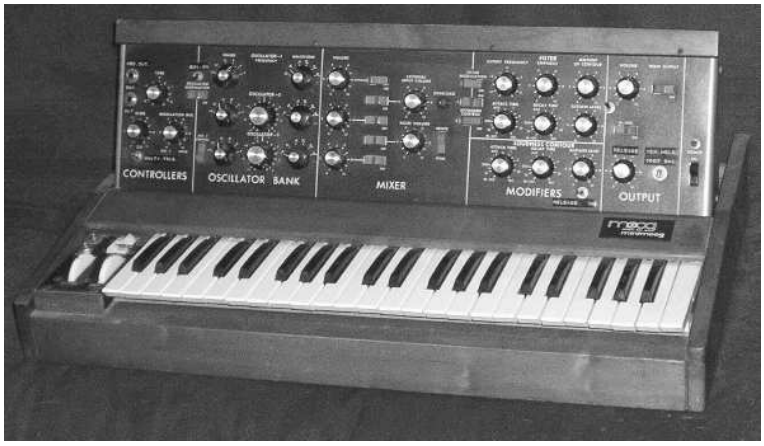


Figure 2.2 The MiniMoog

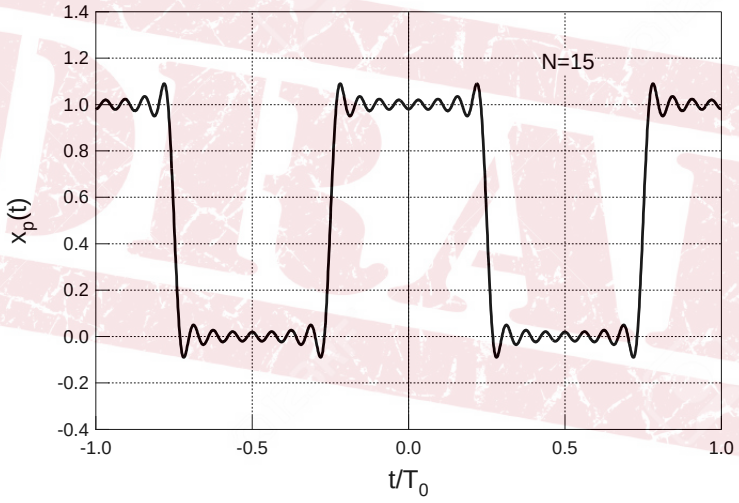


Figure 2.3 Synthesis of a rectangular pulse train with a finite number N=15 of sinusoidal components

Equation (2.1) is the simplest form of a *Fourier series* for a periodic signal. In a sense, the pitfall of such representation is that exact synthesis of the periodic signal theoretically requires an *infinite* number of components. Nonetheless, such a representation can be used in the practice by truncating the series to a (small) number of significant components only, just as the MiniMoog used to do. Figure 2.3 shows how a periodic rectangular pulse train can be synthesized by the superposition of a finite number of elementary sinusoidal components, according to (2.1). Of course, given a certain waveform that we intend to synthesize, the problem is: what are the correct values of A_k and θ_k to be used in our synthesis equation? Giving a response to this question means *analyzing* signal $x_p(t)$ by means of a proper *analysis equation* that we are to find. This is most easily done by resorting to a complex-number representation of the Fourier series. The key to such representation is

Euler's formula for the cosine function:

$$A_k \cos(2\pi k f_0 t + \theta_k) =$$

$$\frac{1}{2}\left[A_k \exp(\jmath 2\pi k f_0 t) \cdot \exp(\jmath\theta) \; + \; A_k \exp(-\jmath 2\pi k f_0 t) \cdot \exp(-\jmath\theta)\right] \tag{2.2}$$

The real-valued oscillating function is decomposed as the sum of two *rotating vectors* on the complex plane. The first one, $\exp(\jmath 2\pi k f_0 t)$ rotates counterclockwise with a frequency f_0 cycles/s (Hz), and the second one, $\exp(-\jmath 2\pi k f_0 t)$, rotates (counterclockwise) at the *negative frequency* $-f_0$ Hz. Teh sun of the two complex rotating vectors gives just the real-valued cosinusoidal oscillation we started from. This complex decomposition entails the introduction of (complex) signal components with negative frequencies. The amplitude and phase spectra of the sinusoid are collapsed into a single complex-valued coefficient $X_k = A_k \exp[j\theta_k]$ (k positive) that is called the *Fourier coefficient* of $x_p(t)$. Elaborating (2.1) with (2.2), we get the following expression of the Fourier series containing the complex Fourier coefficients X_k:

$$x_p(t) = \sum_{k=-\infty}^{\infty} X_k \exp(\jmath 2\pi k f_0 t) \tag{2.3}$$

that is exactly equivalent to the real-valued form (2.1). Finding the amplitude and phase spectra A_k and θ_k is tantamount to finding the k-th Fourier coefficient X_k. With some effort, it is found that the *analysis equation* we were looking for is relatively simple:

$$X_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x_p(t) \exp(-\jmath 2\pi k f_0 t) dt \tag{2.4}$$

Example 2.1

Let us analyze the *pulse train* $x_p(t)$ we tried to synthesize in Fig. 2.3. Its Fourier coefficient is given by

$$X_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x_p(t) \exp(-\jmath 2\pi k f_0 t) dt = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x_p(t) \cos(2\pi k f_0 t) dt \tag{2.5}$$

where we have exploited the even-symmetry of our waveform. Now, considering that $x_p(t)$ is piecewise-constant, we also have

$$X_k = \frac{1}{T_0} \int_{-T_0/4}^{T_0/4} \cos(2\pi k f_0 t) dt = \frac{1}{T_0} \frac{\sin(2\pi k f_0 t)|_{-T_0/4}^{T_0/4}}{2\pi k f_0} =$$

$$= \begin{cases} 1/2 & k = 0 \\ 0 & k = 2m, m \neq 0 \\ \frac{(-1)^m}{\pi k} & k = 2m+1 \end{cases} \tag{2.6}$$

The Fourier coefficients X_k turns out to be real-valued (due to the even symmetry of $x_p(t)$); the resulting amplitude line spectrum of $x_p(t)$ is shown in Fig. 2.4.

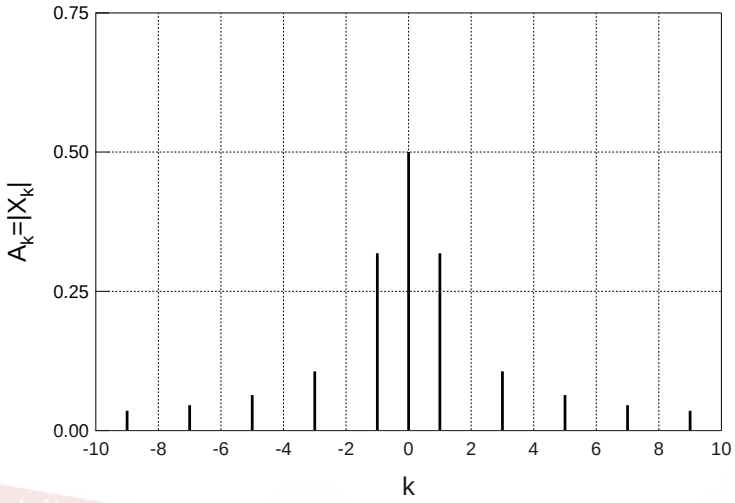


Figure 2.4 Amplitude spectrum of a rectangular pulse train

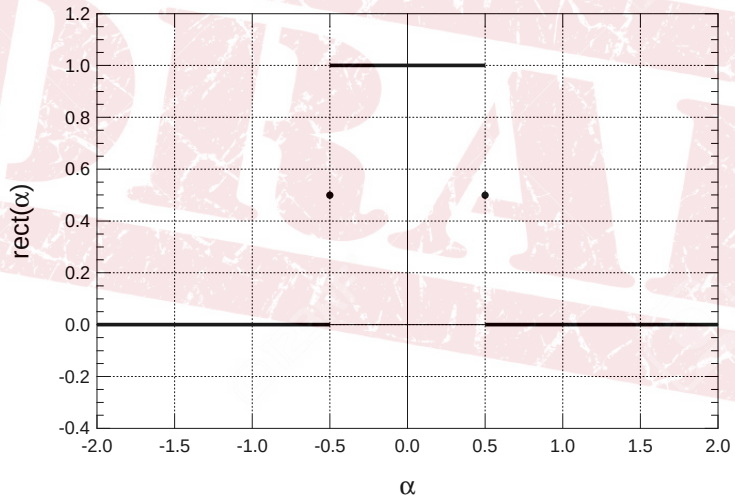


Figure 2.5 rect signal

2.1.2 Non-periodic Signals and the Fourier Transform

What was said until now was only applicable to *periodic* signals. What happens with impulsive signals $x(t)$ that are not periodic? Figure 2.5 shows a *rectangular pulse* that we define as follows:

$$\text{rect}(t/T) \triangleq \begin{cases} 1 & |t| \leq T/2 \\ 1/2 & |t| = T/2 \\ 0 & \text{elsewhere} \end{cases} \tag{2.7}$$

The question is: is this kind of signal amenable to Fourier analysis/synthesis? The answer is (of course) positive, and how to do it can be found quite easily if we think of a non-periodic

signal as a periodic signal with *infinitely-long* repetition period. The fundamental frequency thus becomes vanishingly small (infinitesimal), and so two components in the frequency spectrum of the signal that were previously separated by $\Delta f = (k+1)f_0 - kf_0 = f_0$ now become infinitesimally close to each other. The frequency spectrum of the signal that used to be *discrete* (the line spectrum of Fig. 2.4 with a frequency "quantum" given by the fundamental frequency f_0) now becomes *continuous*. The relevant analysis equation turns out to be now

$$X(f) = \int_{t=-\infty}^{\infty} x(t) \exp(-\jmath 2\pi f t) dt \tag{2.8}$$

where the frequency f that appears as the argument of this complex-valued quantity $X(f)$ takes all *real* values with continuity.

This counterpart of the former Fourier coefficient is called the *Fourier Transform* (FT) of $x(t)$ and bears the same meaning as X_k. It has an amplitude $|X(f)|$ and a phase $\angle X(f)$, so that we still speak of (continuous) amplitude and phase spectra, respectively. Since the spectrum is now continuous, the synthesis equation cannot be a series any more, rather it is expressed in the form of a *Fourier Integral*:

$$x(t) = \int_{f=-\infty}^{\infty} X(f) \exp(\jmath 2\pi f t) dt \tag{2.9}$$

This relation is also called the *Inverse Fourier Transform* (IFT) of $X(f)$. The physical meaning that we can attach to the pair of relations (2.8)-(2.9) is the same as with the Fourier coefficient-series pair: the IFT is a synthesis equation that tells us how to build our own signal starting from a set of simpler components (the complex-valued sinusoids), and the FT tells us the specific values of the amplitude and phase of each sinusoid that has to be used in our synthesis procedure to build a specific signal. We can show easily that the FT $X(f)$ of a real-valued $x(t)$ (that we will denote at times $\mathcal{F}[x(t)]$) has a particular kind of symmetry that is called *Hermitian*: $X(-f) = X^*(f)$, i,e, $|X(-f)| = |X(f)|$, and $\angle X(-f) = -\angle X(f)$.

We will not waste any precious space in describing the many features of the FT as a tool for signal design and analysis in communications engineering. We just want here to recall some elementary results about FT theory that will be used in many places in the Chapters to follow. For instance, it is an easy exercise for the reader to show that the FT of the time-shifted version $x(t - t_0)$ of the signal $x(t)$ is

$$\mathcal{F}[x(t - t_o)] = X(f) \exp(-\jmath 2\pi f t_0) \tag{2.10}$$

so that the amplitude spectrum of the signal is left unchanged, and the phase spectrum is modified by a term proportional to the frequency of each component. Similarly, it is easy to show what happens if we perform an operation of radio-frequency *modulation* on the signal $x(t)$ as follows:

$$x_{RF}(t) = x(t)\cos(2\pi f_0 t) = \frac{x(t)\exp(\jmath 2\pi f_0 t) + x(t)\exp(-\jmath 2\pi f_0 t)}{2} \tag{2.11}$$

where f_0 is the *carrier frequency*. The corresponding modification of the FT is

$$X_{RF}(f) = \frac{X(f + f_0) + X(f - f_0)}{2} \tag{2.12}$$

that is, a frequency-shift of each of the frequency components the modulating signal is made of.

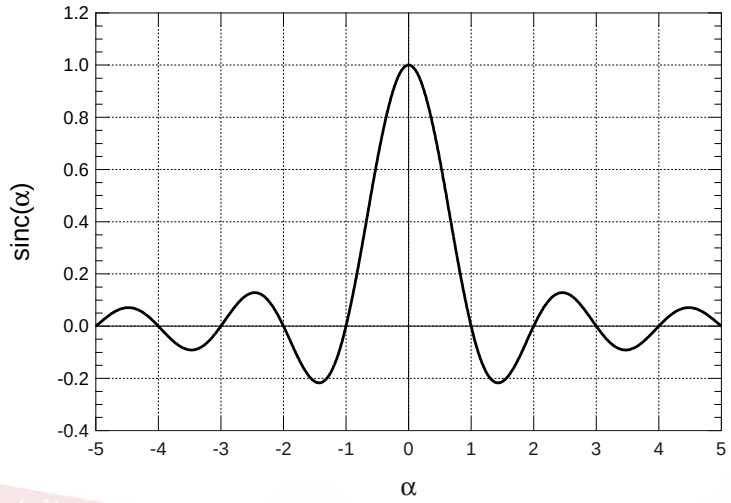


Figure 2.6 sinc function

Example 2.2

Let us find the FT of the rect function $x(t)$ in (2.7):

$$X(f) = \int_{-\infty}^{+\infty} \text{rect}(t) \exp(-\jmath 2\pi f t) dt = \int_{-T/2}^{+T/2} \exp(-\jmath 2\pi f t) dt$$

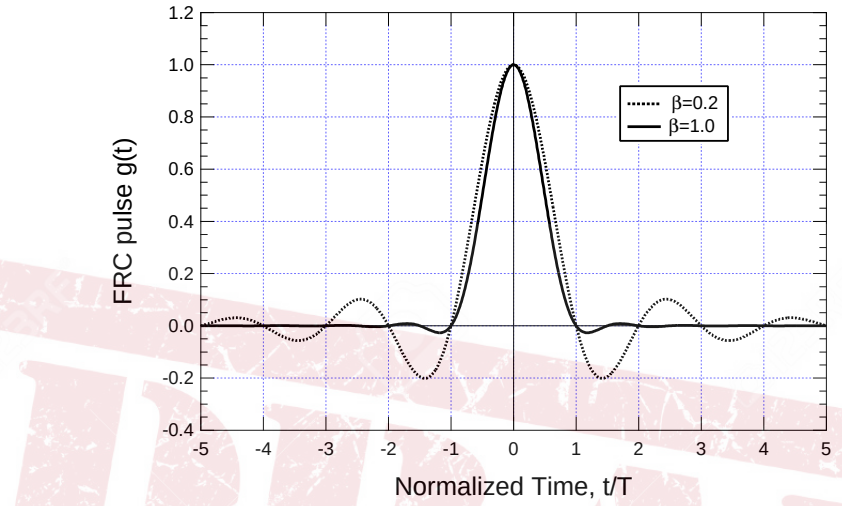
$$= \frac{\sin(\pi f T)}{\pi f} = T \frac{\sin(\pi f T)}{\pi f T} = T\text{sinc}(fT) \tag{2.13}$$

We have introduced here a new identifier for a special waveform that we will extensively use in the following: the so-called sinc function (represented in Fig.2.6) that we define as $\text{sinc}(\alpha) = \sin(\pi\alpha)/(\pi\alpha)$.

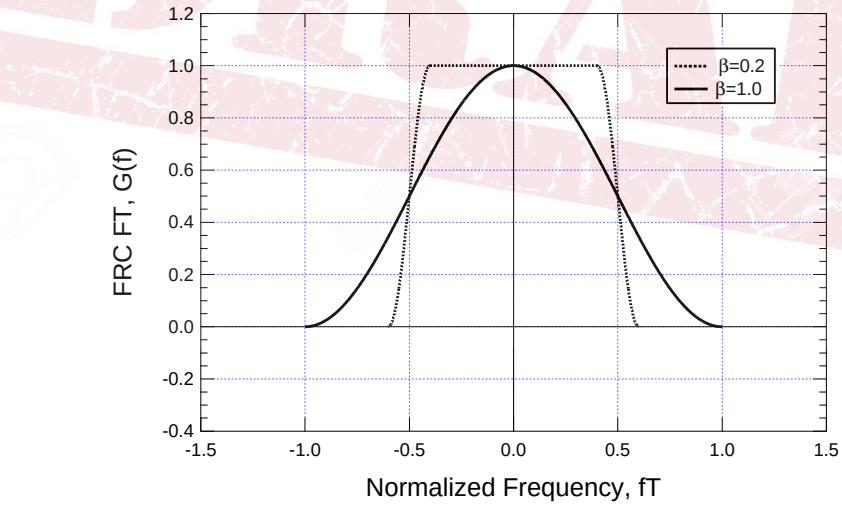
2.1.3 Bandlimited Signals

We will say that a signal is *bandlimited* when it has an amplitude spectrum $|X(f)|$ that is confined into a limited frequency band $[-B, B]$. Such limitation may hold exactly (*strictly* bandlimited signal) or to a good approximation, in the sense that out of the interval $[-B, B]$ the signal components, though not exactly null, are so small as to be considered negligible. An example of a (strictly) bandlimited signal is the popular *Frequency Raised Cosine* (FRC)pulse (or *Nyquist's* pulse) given by

$$g_N(t) = \text{sinc}(t/T) \frac{\cos(\beta\pi t/T)}{1 - (2\beta t/T)^2}$$



(a)



(b)

Figure 2.7 Waveform (a) and Fourier Transform (b) of the FRC pulse

$$G_N(f) = \begin{cases} T & |f| < (1-\beta)/2T \\ \frac{T}{2}\left\{1 + \cos\left[\frac{\pi T}{\beta}\left(|f| - \frac{1-\beta}{2T}\right)\right]\right\} & (1-\beta)/2T \leq |f| \leq (1+\beta)/2T \\ 0 & \text{elsewhere} \end{cases} \tag{2.14}$$

and whose spectrum/waveform are represented in Fig. 2.7. Here, the bandwidth is $B = (1+\beta)/2T$, and β, $0 \leq \beta \leq 1$ is a parameter that regulates the signal bandwidth and that is called *roll-off factor*. The value $1/2T$ is the so-called *Nyquist frequency*, corresponding to the minimum pulse bandwidth when $\beta = 0$.

2.1.4 Dirac's delta function

A peculiar signal that we will often use in the following chapters is Dirac's *delta function* $\delta(t)$. Its name is a little bit defying, since $\delta(t)$ is not actually a signal in the classical sense. We may speak of a *generalized signal* whose definition and existence is only justified through an integral property. Dirac's delta is in fact defined through the so-called *sampling* or *sifting* property:

$$\delta(t) \quad : \quad \int_{-\infty}^{-\infty} x(t)\delta(t)dt = x(0) \tag{2.15}$$

where $x(t)$ is any ordinary signal with no discontinuity at $t = 0$. From (2.15) we immediately have as a particular case

$$\int_{-\infty}^{-\infty} \delta(t)dt = 1 \tag{2.16}$$

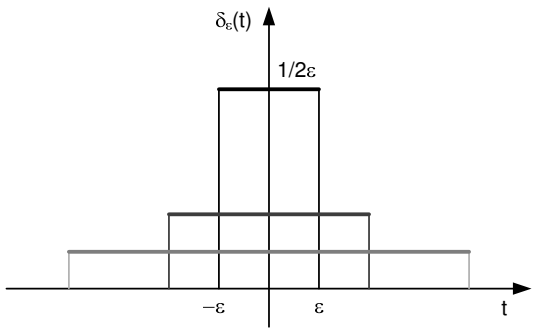
so we may say that the Dirac's function has "unit area". It is easily argued that no ordinary function with property (2.15) exists, but this new mathematical entity proves useful in system theory and linear filtering, as we'll see in a while.

The standard heuristic representation of the delta function (also called *unit impulse*), whose rigorous treatment is found within the so-called *distributions theory*, can be obtained with the aid of a sequence of functions. Assume we have a rect function with duration $T = 2\varepsilon$ and amplitude $A = 1/2\varepsilon$ as represented in Fig. 2.8(a). The "area" of this signal is 1, irrespective of ε. Assume now that this pulse is made shorter and shorter (and consequently, taller and taller) keeping its unit area but becoming thinner and thinner, as suggested in Fig. 2.8(a). The limit of this pulse is a heuristic representation of $\delta(t)$: something whose time width is null, but whose amplitude is infinite, so that its area is unitary. This is what is symbolically depicted in Fig. 2.8(b) as the standard representation of a delta "function". Of course, such a signal does not exist in the ordinary sense.

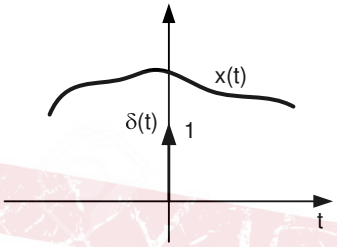
The definition of $\delta(t)$ that follows our heuristic representation is

$$\delta(t) \stackrel{\triangle}{=} \lim_{\varepsilon \to 0} \frac{1}{2\varepsilon} \text{rect}\left(\frac{t}{2\varepsilon}\right) \tag{2.17}$$

This relation not only gives an idea about how the delta function "looks like", but can also be used in the practice, provided that i) $\delta(t)$ appears under an integral operator (as in its definition (2.15)), and ii) the limit in (2.17) is moved *outside* the integral operator, i.e., it is computed *subsequently* to the computation of the integral. The reader may verify (2.16) using this new definition. It is also easy to show that the definite integral of $\delta(t)$ on finite intervals of the kind $\int_b^a \delta(t)dt$ gives a value equal to 1 when the instant $t = 0$ lies within (a, b), otherwise it gives 0.



(a)



(b)

Figure 2.8 Definition of Dirac's function (a), and its symbolic representation (b)

Dirac's delta is also peculiar as far as its FT $\Delta(f)$ is concerned. First, the problem of finding the FT of $\delta(t)$ is well-posed since the FT (2.8) is an integral operator. Second, its computation is trivial, according to (2.15):

$$\Delta(f) = \int_{t=-\infty}^{\infty} \delta(t) \exp(-\jmath 2\pi f t) dt = \exp(-\jmath 2\pi f t)|_{t=0} = 1 \qquad (2.18)$$

The FT of the unit impulse is thus *constant* on all frequencies, with no bandlimitation whatever.

2.2 Linear Filtering

In the following Chapters, we will familiarize with a number of signal processing functions that are implemented in a digital data receiver for wireless communications.

2.2.1 Systems and Signals

The simplest and most fundamental of such operations is perhaps *filtering*. In its simplest realization, filtering means designing a device, an electronic circuit, a piece of software or, in a word, a *system* that changes an *input* signal $x(t)$ into an *output* signal $y(t)$ according to some processing criteria. Our notation to indicate this will be

$$y(t) = \mathcal{T}[x(\alpha); t] \qquad (2.19)$$

where $\mathcal{T}$ is an operator representing the signal processing function performed by the system, and where we indicate that the processing depends in general on the *whole* input waveform

$x(\alpha)$ and on time as well. The simplest family of system are the *linear filters* that obey the *superposition rule*. Assume that we know that

$$y_1(t) = \mathcal{T}[x_1(\alpha); t] \quad , \quad y_2(t) = \mathcal{T}[x_2(\alpha); t] \tag{2.20}$$

and that we build up a signal $x(t)$ as a weighted superposition (i.e., a linear combination) of x_1 and x_2 as $x(t) = \alpha_1 x_1(t) + \alpha_2 x_2(t)$. The system is a *linear* filter iff

$$y(t) = \mathcal{T}[x(\alpha); t] = \mathcal{T}[\alpha_1 x_1(\alpha) + \alpha_2 x_2(\alpha); t] = \alpha_1 y_1(t) + \alpha_2 y_2(t) \tag{2.21}$$

This means that the output $y(t)$ can be obtained as a linear combination with the same coefficients α_1 and α_2 of the "single" outputs of the system to the single inputs x_1 and x_2.

In addition to the property of linearity many filters used in the practice are also *time-invariant*. This means that their behavior does not change with time. Specifically, if we know that $y(t) = \mathcal{T}[x(\alpha); t]$, and we later submit to the system a time-shifted version of the same signal, namely, $x_{TS}(t) \stackrel{\triangle}{=} x(t - t_0)$, we expect that the filter output $y_{TS}(t)$ be just the same waveform that we had earlier, modified only by the same time shift we introduced on the input:

$$y_{TS}(t) = \mathcal{T}[x_{TS}(\alpha); t] = \mathcal{T}[x(\alpha - t_0); t] = y(t - t_0) \tag{2.22}$$

Linear, Time-Invariant (LTI) systems are particularly simple to design and analyze, nonetheless they prove useful in many instances of signal processing, and especially for the detection of known signals embedded into noise.

2.2.2 Time- and Frequency-Characterization of LTI Systems

The properties of any LTI filter are completely characterized by the knowledge of a particular signal that is associated to the system, namely, its *impulse response*. The impulse response of an LTI system is just the system output in response to a $\delta(t)$ input:

$$h(t) \stackrel{\triangle}{=} \mathcal{T}[\delta(\alpha); t] \tag{2.23}$$

In particular, it can be easily shown that the response (output) of an LTI to a generic input $x(t)$ can be found as

$$y(t) = \int_{-\infty}^{\infty} x(\alpha) h(t - \alpha) d\alpha = x(t) \otimes h(t) \tag{2.24}$$

This kind of "mixing" of the two signals x and h to give y deserves a specific name and notation: it is called the *aperiodic convolution* between the two signals, and is denoted by the symbol $\otimes$ as in (2.24). The convolution is a symmetric, associative, and distributive operator, as can be easily proved. The transformation carried out by an LTI system on its input signal $x(t)$ to give the output signal $y(t)$ is often symbolized in a graphical form as in Fig. 2.9 where the impulse response of the system is explicitly indicated.

Fourier analysis of a signal reveals a fundamental tool also in the characterization of the properties of an LTI system. Everything revolves around a cardinal property of convolution. Starting back from the constituent relation (2.24) of an LTI system, and taking the FT of both sides of the equation, we find:

$$Y(f) = X(f) \cdot H(f) \tag{2.25}$$

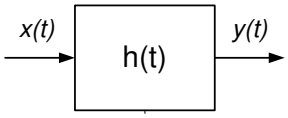


Figure 2.9 Graphical representation of the transformation effected by an LTI system

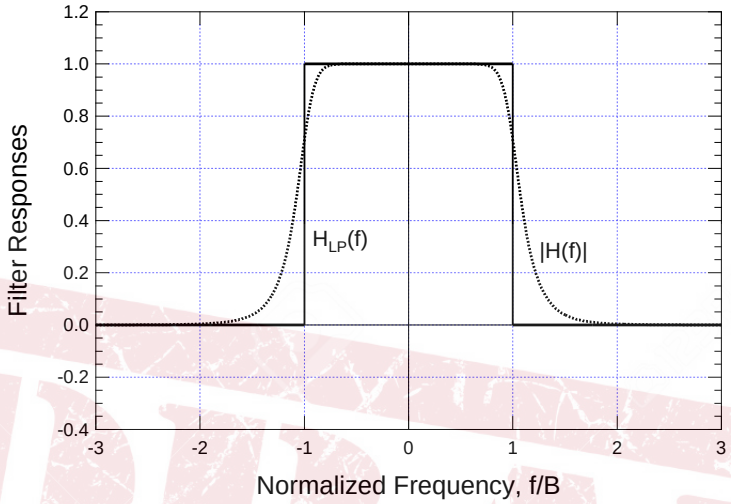


Figure 2.10 Amplitude response of lowpass filters

where $H(f)$ is the FT of the impulse response $h(t)$. It is seen that a (complicated) operation of convolution in the time domain has changed into a (simple) operation of product between two transforms in the frequency domain. This suggest that often the analysis and design of an LTI system is much simpler in the frequency domain than in the time domain due to the simpler operation that the system implicitly carries out on the FT of the input signal. The quantity $H(f)$ is called the *frequency response* of the filter, and is an alternative means to provide a full characterization of the behavior of the system. In particular, it can be shown that the response of the system to a purely sinusoidal input $x(t) = \cos(2\pi f_0 t)$ is just another sinusoidal signal at the same frequency in the form

$$x(t) = \cos(2\pi f_0 t) \quad \Rightarrow \quad y(t) = |H(f_0)| \cdot \cos(2\pi f_0 t + \angle H(f_0)) \tag{2.26}$$

The effect of the filter on the sinusoidal signal is an amplitude change by a factor equal to $|H(f_0)|$ (the *amplitude response* of the system at the frequency f_0), and a phase shift by $\angle H(f_0)$ (the *phase response*). The complex-valued version of (5.48) is

$$x(t) = \exp(\jmath 2\pi f_0 t) \quad \Rightarrow \quad y(t) = H(f_0) \cdot \exp(\jmath 2\pi f_0 t) \tag{2.27}$$

If we change the frequency of the sinusoidal signal, the amplitude and phase responses change, and so our system responds to different components at different frequencies in a different way. This is why $H(f_0)$ is called the frequency response, and also suggests that in general the system bears a *selective* (i.e., unequal) behavior with respect to frequency. Different components in the spectrum of a signals are treated differently. Some may pass substantially unaltered, others may be blocked altogether. An example of such behavior is given by the *low-pass* ideal filter, whose frequency response $H_{LP}(f)$ is represented in Fig.

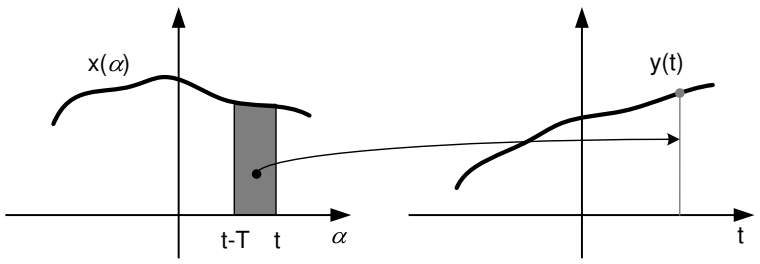


Figure 2.11 Sliding-window Integrator

2.10 (solid line). The "low" frequency components are passed unaltered (they are multiplied by $H_{LP}(f) = 1$), whilst higher-frequency components, in particular those outside the passband B are blocked. The same is substantially true for the system whose amplitude response $|H(f)|$ is also represented in Fig. 2.10 (dashed line). We can not call this an "ideal" filter, but it is nonetheless a good approximation of the ideal filter we have just mentioned, and it is easily realizable.

Example 2.3

We intend to find the impulse and the frequency response of a *sliding window* integrator. This LTI system produces an output whose value at a specific instant is the value of the integral of the input signal on a T-wide integration window immediately leading that instant, as i shown in Fig. 2.11:

$$h(t) = \frac{1}{T}\int_{t-T}^{t} x(\alpha)d\alpha \tag{2.28}$$

We leave to the reader the proof that (2.31) defines indeed an LTI. Assuming this, the impulse response is by definition the output of the system when the input is $\delta(t)$ and is given by

$$y(t) = \frac{1}{T}\int_{t-T}^{t} \delta(\alpha)d\alpha \tag{2.29}$$

As we know by the properties of Dirac's δ function, the integral above is either equal to 1 if the instant $\alpha = 0$ (where $\delta(\alpha)$ is applied) is within the integration interval, otherwise it is equal to 0. This means that the output value is going to be 1 whenever $t - T \leq 0 < t$, that is to say, $0 < t \leq T$, and is going to be 0 outside this interval. The impulse resonse we seek for is a causal rectangular pulse that we can cast into the form

$$h(t) = \frac{1}{T}\text{rect}\left(\frac{t - T/2}{T}\right) \tag{2.30}$$

The frequency response is trivially the Fourier transform of $h(t)$, namely,

$$H(f) = \text{sinc}(fT)\exp(-\jmath\pi fT) \tag{2.31}$$

2.3 Filtering of Random Signals

Random signals or, in the parlance of probability theory, *random processes* play a central role in communications theory. Not only a random process is the standard mathematical representation of all kinds of noise, interference, and disturbance of any sort that may afflict a signal to be detected. A random process is also the representation for the information-bearing signal itself: were the signal known in advance to the receiver, no need of sending it out by the transmitter would arise. The randomness of such signal is just related to the quantity of information that it contains: the more random it looks to the receiver *before* being received, the larger quantity of information it conveys *after* it is actually received.

2.3.1 Basics of random signals

A random signal (process) is a function of time whose shape is not (exactly) known in advance. The value at time t of a deterministic signal $w(t)$ like those encountered until now in the book is exactly known for every possible t, either because we have a mathematical representation of such signal, or because we have recorded it and stored it in a computer file. On the contrary, we can say that the value at a certain time t of a random signal is a *random variable*. The random process is thus the collection of all random variables at all times t, that we denote with the same notation as an ordinary deterministic signal, $w(t)$.

So we do not know in advance the value of the signal for each value of t. Rather, we only know *statistical properties* of such values. When we observe a random signal, what we get is a *realization* of all such random variables time after time, and, after our observation, we are left with a deterministic signal that could not be predicted in advance. The statistical description that we need to completely characterize the properties of such signal may appear to be the probability density function (pdf) $f_w(a;t)$ of the random variable $w(t)$ at time t, that allows to compute probabilities of any kind on $w(t)$. That is not the case indeed. For such signal it may be needed to know, just to make an example, the probability that $w(t) \leq 0$ and, together with this, that after a certain time τ, the probability that $w(t+\tau) \geq 0$. This *joint* probability cannot be computed from $f_w(a;t)$. What we need here is a *second-order* joint pdf $f_w(a_1, a_2; t, t+\tau)$. But then, why stopping just at the second order? The conclusion is that the complete characterization of a random process requires the knowledge of the *class* of joint pdf's of order K of the form

$$f_w(a_1, a_2, \ldots, a_K; t, t+\tau_1, \ldots, t+\tau_{K-1}) \tag{2.32}$$

for *each* (arbitrary large) K. This piece of knowledge is clearly very difficult to get, apart from simple cases. One such exception is that of *Gaussian processes*, for which the pdf's (2.32) are (jointly) Gaussian for any K.

2.3.2 Expectation, Autocorrelation function, and Power Spectral Density

In the practice, a very partial knowledge of the statistical properties of a random process may be sufficient to solve many problems in communications engineering. The simplest statistical property of a random process is its *expectation* function or simply *mean value*:

$$\eta_w(t) \triangleq \int_{-\infty}^{-\infty} a f_w(a;t) da \tag{2.33}$$

This function represents, time instant by time instant, the most likely value that the process is going to assume before it is actually observed. In particular, the relation (2.33) is often

summarized into the following simplified notation:

$$\eta_w(t) = \mathrm{E}\{w(t)\} \tag{2.34}$$

where we introduced the linear operator $\mathrm{E}\{\cdot\}$ called *Expectation* whose definition is self-evident. In many examples of random signals that are dealt with in communications systems (be them information-bearing or just noise), we will see that $\eta_w(t)$ is a constant, and very very often $\eta_w(t) \equiv \eta = 0$.

Example 2.4

We are given the random process

$$w(t) = A\cos(2\pi f_0 t + \Theta) \tag{2.35}$$

where A and f_0 are known, while Θ is a uniform random variable in the interval $[-\pi/2, \pi/2)$. Let us find first the average value $\eta_w(t)$. According to the expectation theorem, instead of using $f_w(a;t)$ to perform the expectation we need, we can use the statistics of the parameter Θ the process depends on:

$$\eta_w(t) = \mathrm{E}\{w(t)\} = E\{A\cos(2\pi f_0 t + \Theta)\}$$
$$= \int_{-\infty}^{-\infty} A\cos(2\pi f_0 t + \theta) f_\Theta(\theta) d\theta = A\int_{-\pi/2}^{\pi/2} \cos(2\pi f_0 t + \theta)\frac{1}{\pi} d\theta$$
$$= -\frac{2A}{\pi}\sin(2\pi f_0 t) \tag{2.36}$$

The average value $\eta_w(t)$ of $w(t)$ is a *first-order* statistical quantity, or first-order statistics. It is computed with a first-order pdf, and gives information of the statistical behavior of our random signal when observed at a *single* time instant. Another example of first-order statistics if the average instantaneous power of the process

$$P_w(t) \stackrel{\triangle}{=} \mathrm{E}\{w^2(t)\} = \int_{-\infty}^{-\infty} a^2 f_w(a;t) da \tag{2.37}$$

The main example of a *second* order statistics is the *autocorrelation function*

$$\begin{aligned} R_w(t,\tau) &\stackrel{\triangle}{=} \mathrm{E}\{w(t)w(t-\tau)\} \\ &= \int_{a_1=\infty}^{-\infty}\int_{a_2=\infty}^{-\infty} a_1 \cdot a_2 f_w(a_1, a_2; t, t-\tau) da_1 da_2 \end{aligned} \tag{2.38}$$

It is computed as the *statistical correlation* between the two random variables $w(t)$ and $w(t-\tau)$ that the random process takes at the two instants t and $t-\tau$, respectively. The autocorrelation function plays a fundamental role in the spectral analysis of a random signal, just as it does for deterministic signals. For deterministic signals, we know that it is a function of the delay τ only, whereas, according to (2.38), $R(t,\tau)$ is a function of both t and τ. We introduce therefore the notion of a wide-sense *stationary* (WSS) process. Assume that

$$R_w(t,\tau) = R_w(\tau) \ , \ \eta_w(t) \equiv \eta_w \tag{2.39}$$

We may say that the properties of the random process are always the same, irrespective of the time instant that we choose as our time origin: the average value does not depend on time, and the autocorrelation function depend only on the *time shift* between the two time instant $t_1 = t$ and $t_2 = t - \tau$ that we consider on our signal. The instantaneous power of a WSS process is constant in time and has the following relation with the autocorrelation function:

$$P_w(t) \equiv P_w = R_w(0) \tag{2.40}$$

For WSS processes, we may define a *power spectral density* (psd) function as the FT of the autocorrelation function:

$$S_w(f) \stackrel{\triangle}{=} \int_{-\infty}^{+\infty} R_w(\tau) \exp(-\jmath 2\pi f\tau) d\tau = 2\int_{0}^{+\infty} R_w(\tau) \cos(2\pi f\tau) d\tau \tag{2.41}$$

where the second equality comes from the fact (easy to show) that $R_w(\tau) = R_w(-\tau)$. The psd function indicates how the power associated to a certain signal is distributed on the different component of the frequency spectrum, and the total signal power can be found as

$$P_w = \int_{-\infty}^{+\infty} S_w(f) df \tag{2.42}$$

As with any random variable, the power of a WSS process can be evaluated as $P_w = \sigma_w^2 + \eta_w^2$.

Example 2.5

Take back into consideration the process we defined in Example 4, but assume that Θ is uniform in $[-\pi, \pi)$. The average value is now easily found to be 0, irrespective of time. We may wonder wether the process is WSS. To possibly verify this, we compute the autocorrelation function $R_w(t, \tau) = \mathrm{E}\{w(t)w(t-\tau)\}$ by virtue of the expectation theorem:

$$\begin{aligned} R_w(t,\tau) &= \mathrm{E}\{\cos(2\pi f_0 t + \Theta)\cos(2\pi f_0(t-\tau) + \Theta)\} \\ &\int_{-\infty}^{-\infty} A\cos(2\pi f_0 t + \theta) A\cos(2\pi f_0(t-\tau) + \theta) f_\Theta(\theta) d\theta \\ &= \frac{A^2}{2\pi}\int_{-\pi}^{\pi} \cos(2\pi f_0 t + \theta)\cos(2\pi f_0(t-\tau) + \theta) d\theta \end{aligned} \tag{2.43}$$

Using trigonometric formulas, it is easy to show that

$$R_w(t,\tau) = \frac{A^2}{2}\cos(2\pi f_0\tau) = R_w(\tau) \tag{2.44}$$

that does *not* depend on t. Our suspicion about the wide-sense stationarity of the process was well founded indeed!

A special example of WSS random signal is the *white noise* process. White light is the one made of all of the colors, we've already mentioned. Color means wavelength, and wavelength means frequency. A white noise process is zero-mean, and such that its psd is *constant throughout the whole frequency range*:

$$S_w(f) = N_0/2 \ \ \forall f \tag{2.45}$$

This is of course an idealization, since no such signal may exist: its associated power is clearly infinite (recall (2.53)). But it is a good model for a random signal that has a much wider bandwidth than that of another "reference" signal or of a system under consideration. This is often the case for the electronic noise (thermal noise, Johnson noise) of active and passive devices, for the "ambient" noise detected by an antennas and so on. In such examples, the noise is generated by the superposition of a multitude of independent elementary components. A well-known result in probability (the central limit theorem) states that the outcome of such superposition is a Gaussian process: any set of random variables $w(t_1)$, $w(t_2)$,...,$w(t_N)$ obtained after observation of the signal at the time instants t_1, t_2,...,t_N has a multivariate Gaussian pdf.

Filtering is a fundamental operation for random signals, just as it is for deterministic waveforms. What happens in particular when a process $w(t)$ is filtered with an LTI system to obtain (the random process) $n(t)$? It is easy to understand how to derive the mean value function. From (2.33) we see that the computation of such function is obtained through *linear* operations: the "Expectation" *operator* is linear. We can invoke thus a "commutative" property of linear operators and say that

$$\begin{aligned}\eta_n(t) = \mathrm{E}\{n(t)\} = \mathrm{E}\{\mathcal{T}[w(\alpha);t]\} = \mathcal{T}[\mathrm{E}\{w(\alpha)\};t] \\ = \mathcal{T}[\eta_w(\alpha);t] = \eta_w(t) \otimes h(t)\end{aligned} \tag{2.46}$$

If $w(t)$ is zero-mean, $n(t)$ will always be zero-mean as well, irrespective of the particular LTI filter we may consider.

Example 2.6

Take back into consideration the parametric process of Example 4

$$w(t) = A\cos(2\pi f_0 t + \Theta) \tag{2.47}$$

where Θ is a uniform random variable in the interval $[-\pi/2, \pi/2)$. Its average value was found to be

$$\eta_w(t) = -\frac{2A}{\pi}\sin(2\pi f_0 t) \tag{2.48}$$

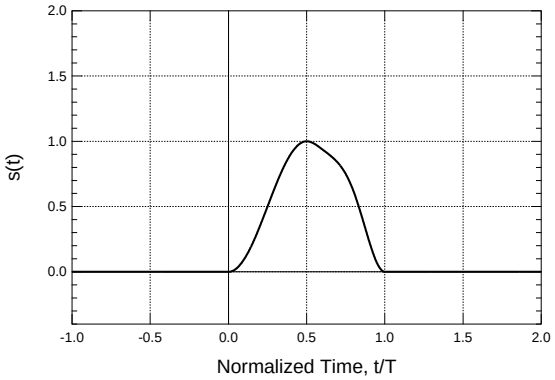
What if $w(t)$ is filtered by the sliding-window integrator of Example 3? Let us call $n(t)$ the result of such filtering. From (2.46) we know that $\eta_n(t) = \eta_w(t) \otimes h(t)$. But $\eta_w(t)$ is *sinusoidal* with time, so that the result of filtering can be easily evaluated through the notion of frequency response of the filter, as in (5.48):

$$\begin{aligned}\eta_n(t) &= -\frac{2A}{\pi}|H(f_0)|\sin(2\pi f_0 t + \angle H(f_0)) \\ &= -\frac{2A\mathrm{sinc}(f_0 T)}{\pi}\sin(2\pi f_0 t + \pi f_0 T)\end{aligned} \tag{2.49}$$

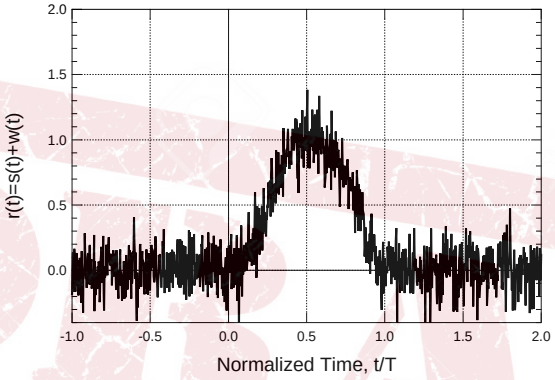
Filtering a WSS random process $w(t)$ is a special case that is easily characterized if we stick to a simple characterization of the output process $n(t)$. We already know from (2.46) how to compute the output average value. The output psd function is also easy to compute since, as happens with deterministic signals,

$$S_n(f) = |H(f)|^2 \cdot S_w(f) \tag{2.50}$$

where $|H(f)|^2$ is the *power response* of the LTI filter.



(a)



(b)

Figure 2.12 Time-limited signal $s(t)$ (a) and its noisy version $r(t)$ (b)

Example 2.7

A certain known signal $s(t)$ is sent out on a wireless radio channel. The receiver collects such signal corrupted by Additive White Gaussian Noise (AWGN) $w(t)$ with psd $N_0/2$ (a sketch of a possible noisy signal is in Fig. 2.12 (b)). The received signal processes $r(t) = s(t) + w(t)$ with an LTI system (filter) to get a filtered signal $y(t) = x(t) + n(t)$, where x and n are the filtered signal and noise components, respectively. Assume also that $s(t)$ is time-limited within the interval $[0, T)$ as in Fig. 2.12 (a), and that the reception filter has impulse response

$$h(t) = \frac{1}{E_s} s(T - t) \tag{2.51}$$

i.e., a reversed and time-delayed (to be causal) version of the transmitted pulse, scaled by the factor $E_s = \int s^2(t)dt$ that is, by the energy of the time-limited signal s. The output of the receive filter at time T is:

$$y(T) = x(T) + n(T) = \frac{1}{E_s} \int_{-\infty}^{+\infty} s(\alpha)s(T - (T - \alpha))d\alpha + N = 1 + N \tag{2.52}$$

where N is a zero-mean Gaussian random variable with variance

$$\sigma_N^2 = \int_{-\infty}^{+\infty} S_n(f)df = \int_{-\infty}^{+\infty} \frac{N_0}{2}|H(f)|^2 df$$

$$= \frac{N_0}{2}\frac{1}{E_s^2}\int_{-\infty}^{+\infty} |S(f)|^2 df = \frac{1}{2E_s/N_0} \tag{2.53}$$

We can compute now the *signal to noise ratio* (SNR) as the ratio between the squared signal component (signal power) and the variance of the noise component (noise power):

$$\text{SNR} \stackrel{\triangle}{=} \frac{x^2(T)}{\sigma_N^2} = \frac{1}{(2E_s/N_0)^{-1}} = \frac{2E_s}{N_0} \tag{2.54}$$

It can be shown that (2.54) is the *best* SNR that can be attained upon filtering of $r(t)$ with *any* LTI system. The shape of $h(t)$ as in 2.51 is the "best match" to the received signal, and so this special filter is called the *matched filter*

2.4 Bandpass Signals and Systems

2.4.1 Baseband equivalent of a bandpass signal

The general form of a sinusoidal bandpass signal at frequency f_0 is

$$x_{BP}(t) = A\cos(2\pi f_0 t + \vartheta) \tag{2.55}$$

where A is the amplitude of the signal and ϑ its phase. An alternative formulation of (2.55) is

$$\begin{aligned} x_{BP}(t) &= A\cos(\vartheta)\cos(2\pi f_0 t) - A\sin(\vartheta)\sin(2\pi f_0 t) \\ &= x_I\cos(2\pi f_0 t) - x_Q\sin(2\pi f_0 t) \\ &= \Re\{(x_I + jx_Q)\exp(j2\pi f_0 t)\} \\ &= \Re\{x_{BB}\exp(j2\pi f_0 t)\} \end{aligned} \tag{2.56}$$

where we have introduced other quantities than the amplitude and phase of the sinusoid, that will be used extensively in the following. First, we defined the *In-phase/Quadrature components* $x_I = A\cos(\vartheta)$ and $x_Q = A\sin(\vartheta)$ as the "projections" of the sinusoid along the two main quadrature carriers at frequency f_0, namely, $\cos(2\pi f_0 t)$ and $-\sin(2\pi f_0 t)$, respectively. Also, we introduced the complex-valued notation of the *baseband equivalent* of our sinusoidal signal $x_{BB} = x_I + jx_Q$. The amplitude of x_{BB} is the amplitude of our sinusoid, and the phase of x_{BB} is its initial phase, $x_{BB} = A\exp(j\vartheta)$. We summarize all this in the easy-to-remember visual representation given in Fig. 2.13

The spectrum of a sinusoid is monochromatic, i.e., it contains only one component at the frequency f_0 (and its twin at $-f_0$ if we use complex-valued FTs). What happens if by virtue of some kind of *modulation* the amplitude and/or phase of $x_{BP}(t)$ are made slowly

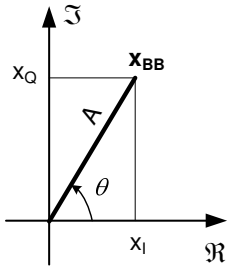


Figure 2.13 Visual representation of I/Q components and amplitude/phase of a baseband signal

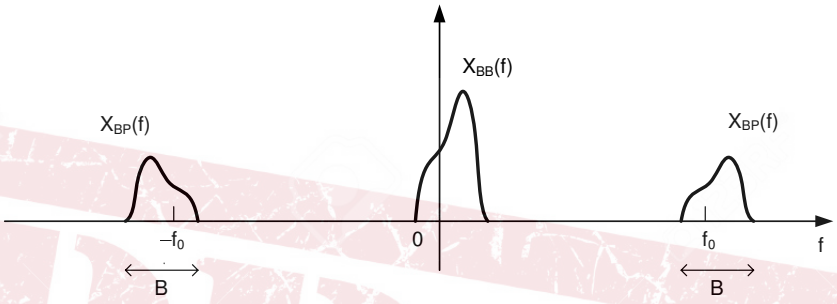


Figure 2.14 Samples of Bandpass and Baseband-equivalent spectra

varying in time? We can write

$$\begin{aligned} x_{BP}(t) &= A(t)\cos(2\pi f_0 t + \vartheta(t)) \\ &= A(t)\cos(\vartheta(t))\cos(2\pi f_0 t) - A(t)\sin(\vartheta(t))\sin(2\pi f_0 t) \\ &= x_I(t)\cos(2\pi f_0 t) - x_Q(t)\sin(2\pi f_0 t) \\ &= \Re\{(x_I(t) + \jmath x_Q(t))\exp(\jmath 2\pi f_0 t)\} \\ &= \Re\{x_{BB}(t)\exp(\jmath 2\pi f_0 t)\} \end{aligned} \tag{2.57}$$

The quantities (I/Q components, baseband equivalent) that we mentioned above are now (slowly) varying in time, where "slowly" is to be intended "on a time scale much larger than $1/f_0$". It turns out that the spectrum of $x_{BP}(t)$ as in (2.57) is no longer monochromatic, but it is concentrated around the *carrier frequency* f_0. The *passband* of such spectrum is $B \ll f_0$, and so the signal is *quasi-monochromatic* or *bandpass*. On the contrary, $x_{BB}(t)$ has a spectrum that is confined to baseband, with a bandwdith B much smaller than f_0. Since the signal is complex valued, the spectrum of $x_{BB}(t)$ will not bear any Hermitian symmetry around 0. Figure 2.14 shows fictional examples of spectra of a bandpass, modulated, quasi-monochromatic signal, as well as its (non-Hermitian-symmetric) baseband equivalent.

2.4.2 The I-Q modulator

So the bandpass signal, once the carrier frequency is known, is completely specified by its baseband equivalent (also called *complex envelope*) $x_{BB}(t)$. Equation (2.57) tells us that $x_{BP}(t) = x_I(t)\cos(2\pi f_0 t) - x_Q(t)\sin(2\pi f_0 t)$. This is not only a mathematical

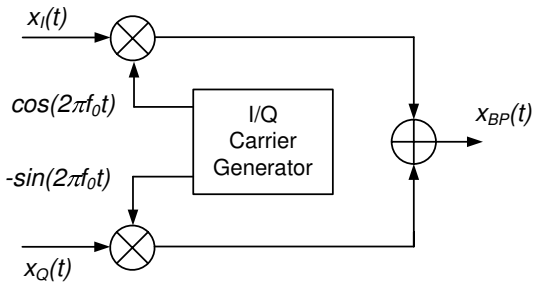


Figure 2.15 General architecture of an I-Q modulator

representation, but it also corresponds to the architecture of the so-called *I-Q modulator* that is used in the practice to generate an arbitrary bandpass modulated signal from its I-Q components as in Fig. 2.15. In the Chapters to follow, we will systematically adopt the complex-valued notation of baseband equivalent signals. To make notation shorter, we will drop the subscripts ${}_BP$ or ${}_BB$ to denote bandpass or baseband signals, respectively, but we will implicitly assume, unless otherwise stated, that all signals are complex envelopes.

2.4.3 The I-Q demodulator

The usual way of modulating a bandpass signal with a digital data stream is to encode the digital information into either $x_I(t)$, or $x_Q(t)$, or both. Once we have done so, and we send the bandpass modulated signal on a physical medium (radio, copper, fiber), the receiver needs to reconstruct either $x_I(t)$, or $x_Q(t)$, or both, to recover (*demodulate*) the digital data. The simplest way to do this starts from the general expression of the bandpass signal:

$$\begin{aligned} x_{BP}(t) &= \Re\{x_{BB}(t)\exp(\jmath 2\pi f_0 t)\} \\ &= \frac{x_{BB}(t)\exp(\jmath 2\pi f_0 t) + x_{BB}^*(t)\exp(-\jmath 2\pi f_0 t)}{2} \end{aligned} \tag{2.58}$$

From this we have,

$$x_{BP}(t)\cdot 2\exp(-\jmath 2\pi f_0 t) = x_{BB}(t) + x_{BB}^*(t)\exp(-\jmath 2\pi\cdot 2 f_0 t) \tag{2.59}$$

and we see that such (complex-valued) signal contains two components: the first one is just the one we intend to get, and the second is something unwanted and centered at the frequency $-2f_0$. What we have to do to get rid of the latter and keep the former is processing this signal with a *lowpass* filter whose bandwidth is just that of $x_{BB}(t)$ to remove the double-frequency components at $2f_0$. The result of this reasoning is simple:

$$x_{BB}(t) = x_I(t) + \jmath x_Q(t) = \{x_{BP}(t)\cdot 2\exp(-\jmath 2\pi f_0 t)\} \otimes h_{LP}(t) \tag{2.60}$$

where $h_{LP}(t)$ is the impulse response of the lowpass filter. Again, (2.60) is not just mathematics, but it is the outline of the so-called *I-Q demodulator* represented in Fig. 2.16 that is implemented in the vast majority of modern radio receivers. It is seen that the product of the received bandpass signal $x_{BP}(t)$ with the complex oscillation $2\exp(-\jmath 2\pi f_0 t)$ is implemented as a pair of real products between the former and the real-imaginary components of the latter, and the lowpass filter is applied to both components as well.

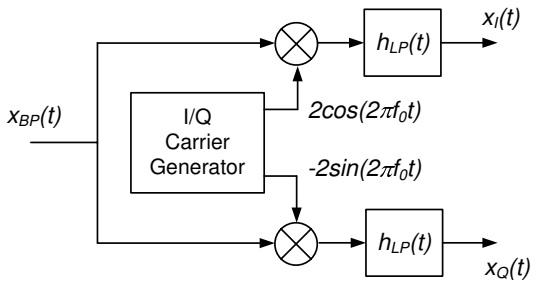


Figure 2.16 General architecture of an I-Q demodulator

2.5 Fourier analysis of digital signals

2.5.1 Analog and Digital signals

In the previous section we have reviewed the main concepts and results in Fourier analysis of time-continuous (*analog*) signals. The same results can be extended to *digital* signals, or, properly speaking, time-discrete ones. The most immediate way of generating a digital signal is using an analog-to-digital converter (ADC), or, in the parlance of signal analysis, performing the operation of *sampling*. Sampling an analog signal $x(t)$ with a certain sampling rate f_s samples/s (or simply Hz) or equivalently a sampling period $T_s = 1/f_s$ means extracting from $x(t)$ a sequence of *samples* $x[n]$ such as

$$x[n] = x(nT_s) \tag{2.61}$$

The square brackets indicates the the time index n they enclose is discrete, as opposed to continuous time t that is usually enclosed into round brackets. The value at time n of $x[n]$ is real-valued, and so its representation theoretically requires an infinite number of digits. In the practice, the ADC represents each value as an integer on a fixed (finite) number of binary digits. This introduces a (small) representation error: what we get out of the ADC is actually $x_q[n] = x[n] + q[n]$, where x_q is the *quantized* version of $x[n]$, and $q[n]$ is the *quantization noise*. When the number of bits in the digital representation of $x_q[n]$ is large (say, larger than 16), the quantization noise can be safely disregarded.

2.5.2 FT of Digital signals

Sampling and the ADC are the foundation of Digital Signal Processing (DSP). DSP techniques are again heavily based on Fourier analysis, so that we have to review the basics of Fourier transforms for time-discrete signals.

Generalizing the notions already introduced for analog signals, it can be easily shown that a non-periodic sequence $x[n]$ can be Fourier-decomposed as

$$x[n] = \frac{1}{f_s} \int_{-f_s/2}^{+f_s/2} \overline{X}(f) \exp(\jmath 2\pi n f / f_s) df \tag{2.62}$$

where $\overline{X}(f)$ is the FT of the sequence $x[n]$. Equation (2.62) is apparently a *synthesis* equation much similar to (2.8) for analog signals. The corresponding *analysis* equation that

gives the FT $\overline{X}(f)$ is

$$\overline{X}(f) = \sum_{n=-\infty}^{+\infty} x[n] \exp(-\jmath 2\pi n f / f_s) \tag{2.63}$$

A fundamental difference exists between the FT of analog and of digital signals. It is seen from (2.62) that the digital signal can be synthesized from a *finite* interval of continuous frequency components, whilst the analog signal requires component at *all* frequencies on the real axis. The reason for this is that the FT $\overline{X}(f)$ of a sequence is a *periodic* function of frequency, with a period equal to the sampling frequency f_s. Thus, the only independent components of $x[n]$ actually lie on an f_s-wide frequency interval, and no more components than those are required in the synthesis. This has also something to do with the property of sampled sinusoids. The sequence extracted by sampling a sinusoidal signal $x_1(t)$ at frequency f_1 is

$$x_1[n] = x_1(n/f_s) = \cos(2\pi n f_1 / f_s) \tag{2.64}$$

The ratio f_1/f_s is sometimes indicated with F_1 and is called the *normalized* frequency. Assume now we have a second sinusoidal signal $x_2(t)$ at the frequency $f_1 + f_s$. $x_2(t)$ is clearly (much) different from $x_1(t)$. After sampling $x_2(t)$ we get

$$x_2[n] = \cos(2\pi n f_2 / f_s) = \cos(2\pi n f_1 / f_s + 2\pi n) = \cos(2\pi n f_1 / f_s) = x_1[n] \tag{2.65}$$

After sampling, the two previously different sinusoidal signals look exactly the same! This means that in the digital domain, there can be no more independent sinusoidal components to synthesize a signal than those into a f_s-wide "base" interval: a component outside that interval is actually the "image" of another one that lies into the base interval at a distance equal to an integer multiple of f_s.

Example 2.8

Assume that we sample a time-continuous exponential signal $x(t) = \exp(-t/\alpha)\mathrm{u}(t)$ with a sampling interval $T - s$. What we get is

$$x[n] = x(nT_s) = \exp(-nT/\alpha) = a^n \mathrm{u}[n] \tag{2.66}$$

where $a \triangleq \exp(-T/\alpha) < 1$ is a real constant depending on the time constant of the signal and on the sampling frequency. The FT of the resulting sequence is

$$\overline{X}(f) = \sum x[n] \exp\left(-\jmath 2\pi n f T_s\right) = \sum_{n=0}^{\infty} a^n \exp\left(-\jmath 2\pi n f T_s\right)$$

$$= \sum_{n=0}^{\infty} \left[a \exp\left(-\jmath 2\pi f T_s\right)\right]^n = \frac{1}{1 - a \exp\left(-\jmath 2\pi f T_s\right)} \tag{2.67}$$

where certainty about the convergence of the series comes from the condition

$$\left|a \exp\left(-\jmath 2\pi f T_s\right)\right| = |a| < 1$$

The amplitude and phase spectra of $x[n]$ as resulting from (2.67) are shown in Fig. 2.17. Note that they are shown across a frequency span equal to $\pm f_s/2$ since the two functions are *periodic* with period f_s.

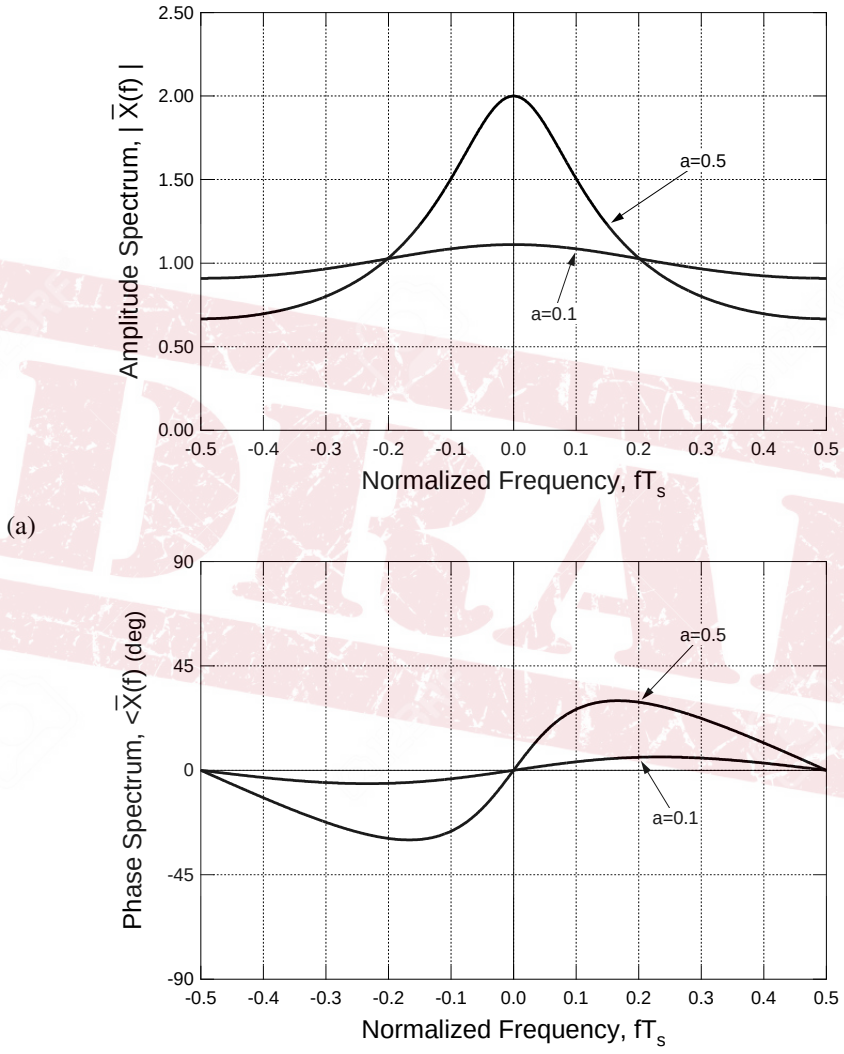


Figure 2.17 Amplitude (a) and phase (b) spectrum of the exponential sequence

Many of the properties that we already mentioned for the FTs of analog signals hold true for FT of sequences as well, whit minor modifications. We refer in particular to Hermitian symmetry, delay and modulation theorems, and so on.

Example 2.9

In the domain of analog signals, special attention was devoted to the definition and properties of Dirac's delta function (Sect. 2.1.4). Is there something similar in the digital domain? The answer is simpler than expected. While $\delta(t)$ was a very special signal, its time-discrete counterpart is just the ordinary sequence

$$\delta[n] \triangleq \begin{cases} 1 & n = 0 \\ 0 & \text{elsewhere} \end{cases} \tag{2.68}$$

Its FT is clearly

$$\overline{\Delta}(f) = \sum_{n=-\infty}^{+\infty} \delta[n] \exp(-\jmath 2\pi n f / f_s) = 1 \tag{2.69}$$

just like the FT of $\delta(t)$.

If our sequence $x[n]$ comes from sampling of an analog signal $x(t)$ with FT $X(f)$, a fundamental question is: what is the relation between the FT $X(f)$ of the signal we start from, and the FT of the resulting sequence $x[n]$? The answer is called *Poisson's relation* and reads

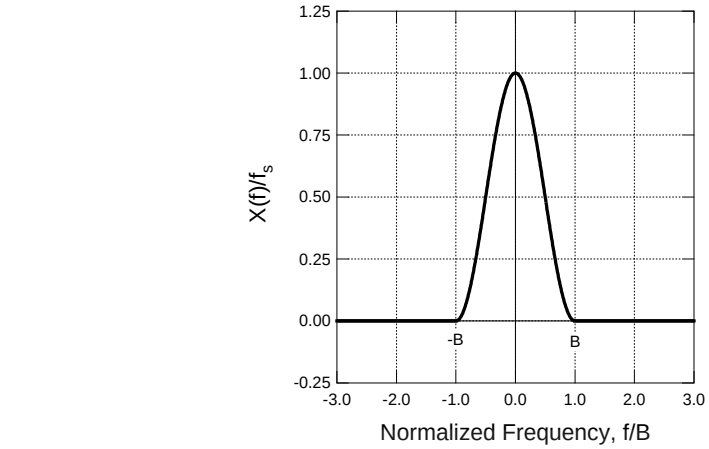
$$\overline{X}(f) = f_s \sum_{k=-\infty}^{+\infty} X(f - k f_s) \tag{2.70}$$

This equation tells us that the FT of $x[n]$ is the superposition of an infinite series of repetitions of the original FT of the analog signal, with a repetition period that is equal to the sampling frequency f_s. This of course gives a periodic FT $\overline{X}(f)$ as that of any digital signal. Notice that there might a sort of "interference" between adjacent repetitions of the original spectrum that is called *aliasing*. An example of aliasing due to sampling is shown in Fig. 2.18 (b) that show the FT $\overline{X}(f)$ of the sequence $x[n]$ obtained after sampling thh analog signal $x(t)$ whose FT $X(f)$ is hown in Fig. 2.18 (a). Such superposition of adjacent spectra does not occur only if the original signal $x(t)$ is bandlimited into the band B, and the sampling frequency is larger than $2B$. The condition $f_s > 2B$ is called the *Nyquist's* condition. When it is verified, there's no aliasing in the sampled signal spectrum. Compare in this respect Fig. 2.18 (a) showing again $\overline{X}(f)$ resulting form the sampling of the signal in Fig. 2.18 (a), this time meeting Nyquist's condition. In particular, the main replica with $k = 0$ in Poisson's relation (2.70) that lies in the main interval $[-f_s/2, f_s/2)$ of the FT is a perfect replica (apart from an immaterial scale factor) of the original spectrum $X(f)$.

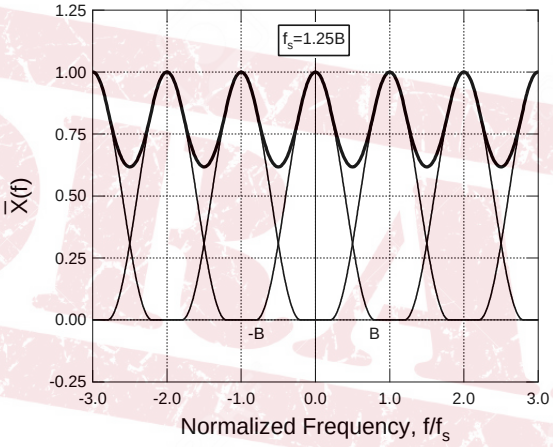
2.5.3 Filtering and Interpolation of digital signal

Needless to say, the notion of an LTI system translates nicely and neatly into the digital domain as well. Using a notation similar to the one we introduced for analog systems, a digital LTI filter is identified by an impulse response

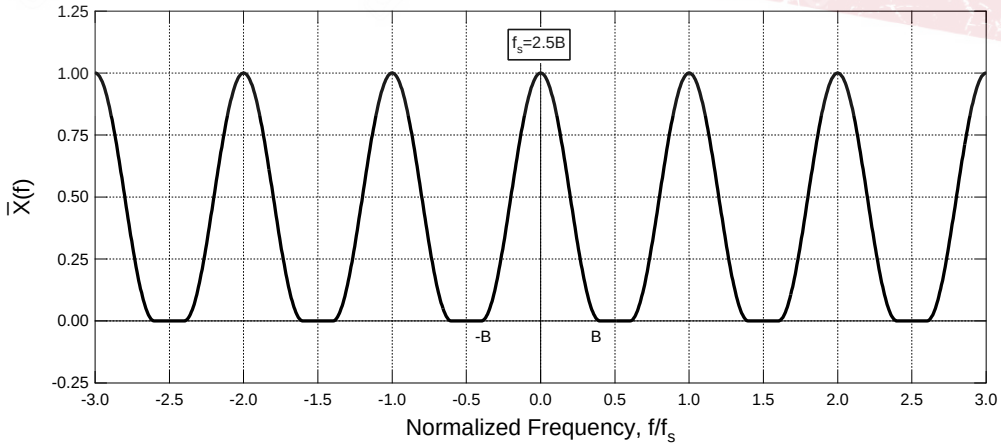
$$h[n] \triangleq \mathcal{T}[\delta[n]; n] \tag{2.71}$$



(a)



(b)



(c)

Figure 2.18 Spectrum of an analog signal (a), of the sampled digital signal with aliasing (b), and of the sampled digital signal without aliasing (c)

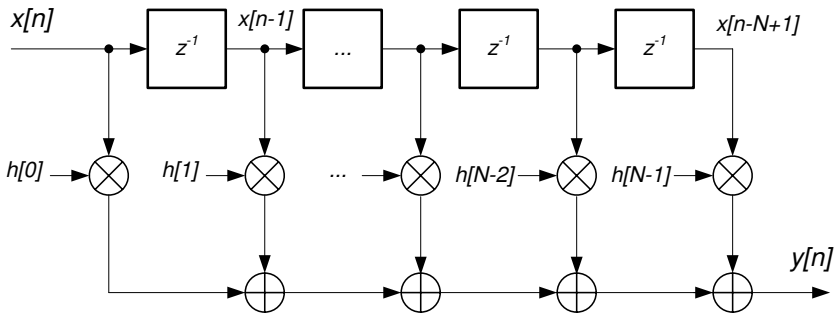


Figure 2.19 Implementation of an FIR filter

and its operation is described by the time-discrete aperiodic convolution between the input signal $x[n]$ and such impulse response $h[n]$:

$$y[n] = x[n] \otimes h[n] \stackrel{\triangle}{=} \sum_{m=-\infty}^{+\infty} x[m] \cdot h[n-m] = \sum_{m=-\infty}^{+\infty} h[m] \cdot x[n-m] \tag{2.72}$$

The system may also have a frequency response

$$\overline{H}(f) \stackrel{\triangle}{=} \sum_{n=-\infty}^{+\infty} h[n] \exp(-\jmath 2\pi n f / f_s) \tag{2.73}$$

so that the frequency-domain input-output relationship still is

$$\overline{Y}(f) = \overline{X}(f)\overline{H}(f) \tag{2.74}$$

If the impulse response $h[n]$ of the LTI system has a finite number of samples different from zero, the filter is *Finite Impulse Response* (FIR), otherwise it is IIR (*Infinite Impulse Response*). The operations to be computed to implement an FIR filter can be represented as in Fig. 2.19, where the blocks labeled z^{-1} implement the delay of their input sequence by one sample (unit-delay element), and where we have assumed that $h[n] = 0$ when $n < 0$ or $n \geq N$.

Once we have turned an analog signal into a digital sequence via sampling, and after we have possibly performed some digital processing on such (digital) signal (even simple storage on a digital medium such as a Compact Disc or a flash memory stick), it may be desired to reconstruct an analog signal form the resulting (retrieved) sequence. This reverse-sampling operation is called *interpolation* and in the practice it is implemented by a *Digital to Analog Converter* (DAC). The general form of an interpolated signal $\hat{x}(t)$ is

$$\hat{x}(t) = \sum_{n=-\infty}^{+\infty} x[n] \cdot p(t - nT_s) \tag{2.75}$$

where $x[n]$ is the sequence being interpolated, and $p(t)$ is the pulse shape that is specific of a particular interpolator. If $p(t)$ is the rectangular pulse in Fig. 2.20 (a), then we have a zero-hold interpolator that basically produces a sample-and-hold signal. If on the contrary $p(t)$ is the triangular pulse in Fig. 2.20 (b) we get a linear intepolator that joins consecutive

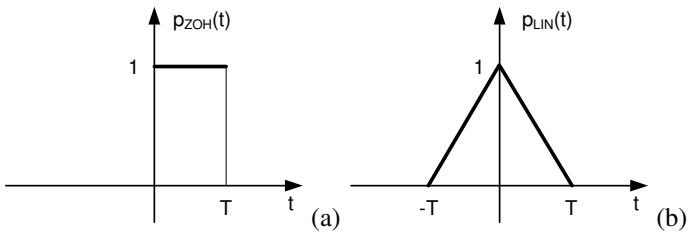


Figure 2.20 Interpolating pulses: (a) ZOH interpolator; (b) linear interpolator

values of the signal samples with straight line segments to produce the interpolated analog signal $\hat{x}(t)$. The frequency-domain counterpart of (2.75) is very simple:

$$\hat{X}(f) = P(f)\overline{X}(f) \tag{2.76}$$

Notice that both $\hat{X}(f)$ and $P(f)$ are FTs of analog signals, whilst $\overline{X}(f)$ is the FT of the sequence $x[n]$ to be interpolated.

2.5.4 The sampling theorem

A rather fundamental question about signal sampling comes immediately to one's mind. Once an analog signal is sampled, and its samples are all collected and, say, stored, is it possible to fully recover such signal with no loss? At first sight the response is NO, since when converting a signal from time-continuous to time-discrete all that is in between samples appears to have been lost for ever. BUT... a glance in the frequency domain may give more hope. If the signal is bandlimited to B and we meet the Nyquist's condition $f_s \geq 2B$, we already know that we have no aliasing, and we "see" an undistorted replica of the spectrum of the analog signal in the spectrum of our sequence (the replica with $k = 0$ in the Poisson formula (2.70)). The real issue is how to recover such replica and get back to the analog domain. The answer is relatively simple: we are to use an appropriate interpolator that preserves the replica with $k = 0$ while canceling all of the others. Figure 2.21 explains that (the reference is again the analog signal spectrum shown in Fig. 2.18 (a)): we need an interpolator whose FT $P(f)$ is flat within the frequency interval $[-f_s/2, f_s/2)$ that contains the main replica with $k = 0$, and zero outside that band. Also, it has to compensate the factor f_s in Posisson's relation. In a word, we have to choose

$$P(f) = \frac{1}{f_s}\text{rect}\left(\frac{f}{f_s}\right) = T_s\text{rect}(fT_s) \tag{2.77}$$

Under Nyquist's condition and using this interpolator, it is apparent that $\hat{X}(f) = X(f)$, so that we can say that the issue of reconstructing the sampled signal is now solved. The interpolating pulse that corresponds to such $P(f)$ is trivially $p(t) = \text{sinc}(t/T_s)$, so that the relevant interpolation formula is

$$\hat{x}(t) = \sum_{n=-\infty}^{+\infty} x[n] \cdot \text{sinc}\left(\frac{t - nT_s}{T_s}\right) \tag{2.78}$$

that is called the *cardinal interpolator*. Since we known that $\hat{X}(f) = X(f)$, it is also clear that $\hat{x}(t) = x(t)$

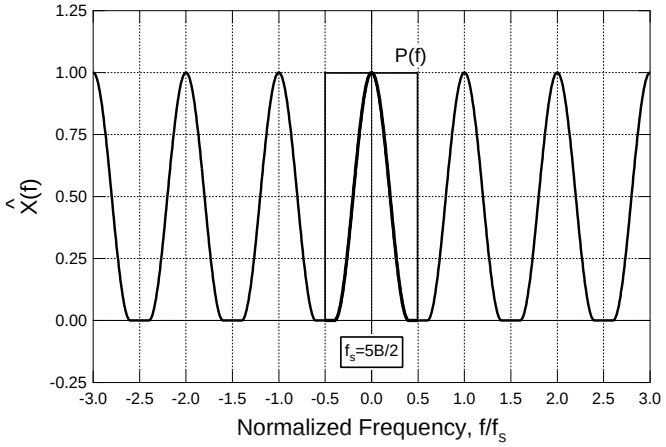


Figure 2.21 Frequency-domain interpretation of cardinal interpolation

2.6 Baseband Transmission of Digital Signals

2.6.1 NRZ and bandlimited digital signals

The basic form of a baseband data signal that conveys digital information (as is encountered on wireline data communications) is that of a plain binary No Return to Zero (NRZ) digital signal:

$$x(t) = A \sum_{n=-\infty}^{+\infty} a[n]g(t - nT) \tag{2.79}$$

Here, A is the overall signal amplitude, $g(t)$ is a unit-amplitude rectangular pulse with time width T, $g(t) = \text{rect}[(t - T/2)/T)]$, and $a[n]$ is the sequence of digital data. For binary *antipodal* signals, $a[n]$ takes either the value -1 or +1, according to the particular sequence (*pattern*) of data to be transmitted. The values -1 and +1 are associated to a logical values 0 or 1, respectively, of the data bits. If the width of the data pulse $g(t)$ is smaller than T, the format of the data signal is called RZ (return to Zero) since the data pulse for each bit falls back to 0 before the next bit is sent out. It is apparent that the signal (2.79) is a repetition of pulses with different amplitude (polarity). The repetition rate $R \stackrel{\triangle}{=} 1/T$ is called the *signaling rate*, *symbol rate* or *clock rate* and is measured in *symbols/s* or *Baud*. The data signal can be thought of as the result of an interpolation as in (2.75) of a digital sequence $a[n]$ coming with a sampling frequency R, with a basic interpolation pulse equal to the data pulse $g(t) = p_{ZOH(t)}$. The resulting signal format used to be called *Pulse-Amplitude Modulation* (PAM).

From the standpoint of the receiver, the specific value at time n of the n-th data symbol $a[n]$ is not known in advance (otherwise there would be no need to carry out any transmission). Therefore, the data signal is modeled as a random process and as such it has to be treated with the tools and approaches that we introduced in Sect. 2.3. The simplest assumption, that is actually well verified in the practice, is that $a[n]$ is a sequence of i.i.d. (independent identically distributed) random variables. In most cases, the values that $a[n]$ can take are also equiprobable.

Although we have implicitly assumed so in our first discussion of digital signals, it is not necessarily true that the data symbols $a[n]$ are binary. This is the reason why we call them "symbols" and not "bits". The symbols of the digital signals that are used in ISDN twisted pairs connections are *quaternary*, i.e., they can take the values $\{-3, -1, +1, +3\}$. Such values are selected according to a law that associates them to pairs of bits to be sent out, for instance $00 \rightarrow -3$, $01 \rightarrow -1$, $11 \rightarrow 1$, $10 \rightarrow 3$ (the so-called *mapping*). It is apparent that in this example a single data symbols carries the information of *two* bits at a time. Therefore, the *bit rate* R_b in the transmission (as measured in bit/s) is different from the symbols rate: $R_b = 2R$. In general, if the symbol $a[n]$ can take one of M values (that is, $a[n]$ belongs to an M-ary *alphabet*), the bit rate is $R_b = \log_2(M) \cdot R$. The simplest alphabet for M-PAM signals is

$$\mathcal{A} \equiv \{-(M-1), -(M-3), \ldots, -1, +1, \ldots, (M-3), (M-1)\} \tag{2.80}$$

NRZ signals are the most elementary form of digital signals, but they are not used in telecommunications (they are only used for data communications in computer networks or data acquistion/storage equipment). The reason for this is related to the *bandwidth* of such signals.

2.6.2 Signals and spectra

How a data signal look like in the time domain, we already saw in the previous section. What is still to be understood is its spectral appearance and bandwidth occupancy. We find first the autocorrelation function of the baseband data signal (2.79):

$$\begin{aligned} R_x(t,\tau) &= E\{x(t)x(t-\tau)\} \\ &= A^2 E\left\{ \sum_{n=-\infty}^{+\infty} a[n]g(t-nT) \sum_{m=-\infty}^{+\infty} a[m]g(t-\tau-mT) \right\} \\ &= A^2 \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} E\{a[n]a[m]\}\, g(t-nT)g(t-\tau-mT) \\ &= A^2 \cdot A_2 \sum_{n=-\infty}^{+\infty} g(t-nT)g(t-\tau-nT) \end{aligned} \tag{2.81}$$

where A_2 is the average power of $a[n]$, $A_2 = E\left\{a^2[n]\right\}$, and where we took into account that $a[n]$ and $a[m]$ are zero mean and uncorrelated when $n \neq m$. We see that the data signal is not a WSS process, since the autocorrelation function still depends on t and not on τ only. We also see that $R_x(t,\tau)$ is a periodic function of the time t for any time lag τ. When this happens, the process is said to be *cyclostationary*. The psd function of such process can be found by introducing the time-averaged autocorrelation function, where averaging is performed over the repetition period:

$$\rho_x(\tau) \triangleq \frac{1}{T} \int_{t=-T/2}^{T/2} R_x(t,\tau)dt \tag{2.82}$$

The psd function is now defined as the customary FT of $\rho_x(\tau)$. If we take back into consideration (2.81), we get

$$\begin{aligned}\rho_x(\tau) &\triangleq \frac{A^2}{T}\int_{t=-T/2}^{T/2} A_2 \sum_{n=-\infty}^{+\infty} g(t-nT)g(t-\tau-nT)dt \\ &= \frac{A^2\cdot A_2}{T}\sum_{n=-\infty}^{+\infty}\int_{t=-T/2}^{T/2} g(t-nT)g(t-\tau-nT)dt \\ &= \frac{A^2\cdot A_2}{T}\int_{-\infty}^{+\infty} g(t)g(t-\tau)dt \end{aligned} \tag{2.83}$$

We see that the (averaged) autocorrelation function of the random data signal $x(t)$ is proportional to the autocorrelation function of the deterministic signal $g(t)$. Taking the FT of $\rho_x(\tau)$ we get

$$S_x(f) = \frac{A^2\cdot A_2}{T}|G(f)|^2 \tag{2.84}$$

The final result of our long but instructive computation is that the psd function of a data signal is determined by the (squared) amplitude spectrum of the basic data pulse. As a consequence, the bandwidth occupancy of a digital signal is equal to the bandwidth of its elementary pulse. An NRZ signal with a full response rectangular pulse has a psd function

$$S_x(f) = A^2\cdot A_2 T\text{sinc}^2(fT) \tag{2.85}$$

as represented in Fig. 2.22, whose bandwidth is infinite. A practical measure of its spectrum occupancy is the so-called *bandwidth at the first null* that is equal to $1/T$. On the other hand, we know that each physical medium (copper wire, optical fibre, radio band) has a bandwidth limitation. So the real problem in telecommunications is finding a "good", i.e., one that can support a high data rate R_b data signal when sent onto a bandlimited physical channel with bandwidth B.

Filtering an NRZ signal is not efficient, since bandlimiting causes distortion and bad reception of data. The solution is resorting to an *intrinsically bandlimited* data signal. This can be done by using a bandlimited pulse $g(t)$ such as the Nyquist FRC pulse $g_N(t)$ (2.14), whose bandwidth is $B = R(1+\beta)/2$. The most popular bandlimited pulse shape is the *Square-Root Frequency Raised Cosine* (SRFRC) pulse whose FT is proportional to the square root of the FT of the Nyquist FRC pulse $g_N(t)$:

$$G(f) = T\sqrt{G_N(f)/T} = \sqrt{TG_N(f)} \tag{2.86}$$

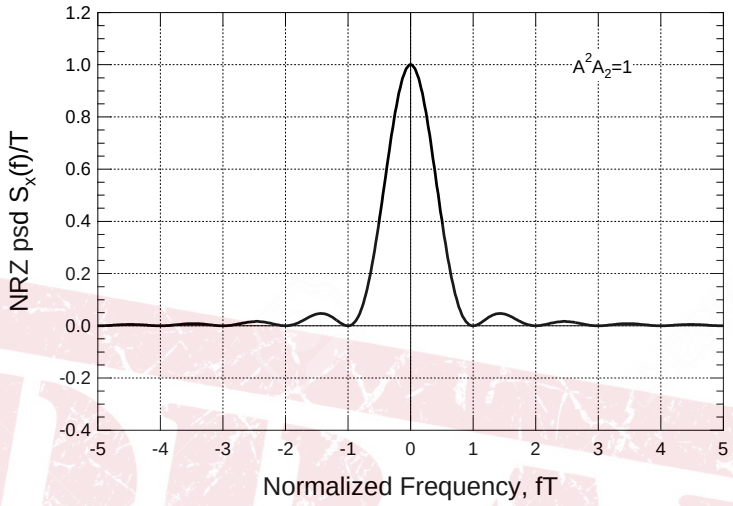
whose waveform is

$$g(t) = \frac{\sin\left(\pi(1-\beta)\frac{t}{T}\right) + 4\beta\frac{t}{T}\cos\left(\pi(1+\beta)\frac{t}{T}\right)}{\pi\frac{t}{T}\left[1-16\beta^2\left(\frac{t}{T}\right)^2\right]} \tag{2.87}$$

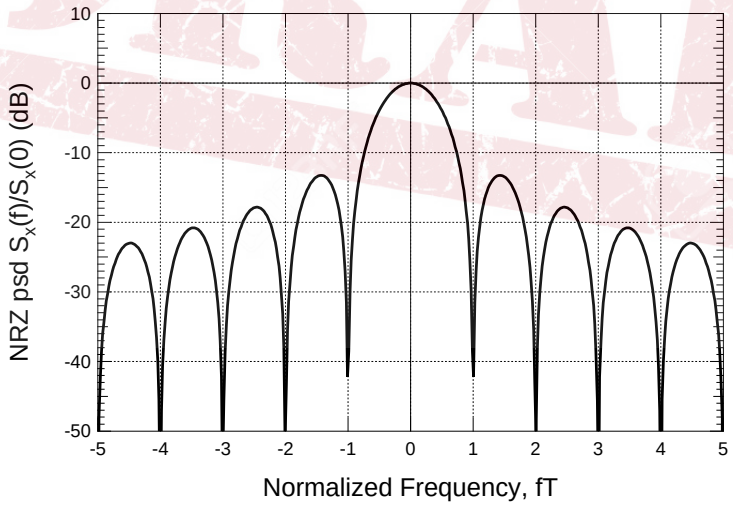
and whose bandwidth is again equal to $R(1+\beta)/2$.

For any kind of basic pulse, integration of (2.84) gives the total power of the data signal as

$$P_x = \frac{A^2\cdot A_2}{T}\int_{-\infty}^{+\infty}|G(f)|^2df = \frac{A^2\cdot A_2}{T}\int_{-\infty}^{+\infty}|g(t)|^2dt = A^2\cdot A_2E_g/T \tag{2.88}$$



(a)



(b)

Figure 2.22 Power spectrum of an NRZ data signal. Linear scale (a) and log (dB) scale (b)

where E_g is the energy of the data pulse $g(t)$. In many cases, for instance the rectangular pulse of NRZ signaling, or the SRFRC pulse, the energy of the pulse turn out to be $E_g = T$, so that $P_x = A^2 \cdot A_2$. For a multilevel PAM signal with M-ary symbols in the alphabet $\{-(M-1), -(M-3), \ldots, -1, +1, \ldots, (M-3), (M-1)\}$ we get $A_2 = (M^2-1)/3$.

2.6.3 Detection of a digital signal on the AWGN channel

One of the fundamental problems in communications theory is the reliable detection of digital data signals in the presence of random noise. The simplest such case is detection in the AWGN channel that is typical of wireless communications over a line-of-sight link (as in satellite communications). We outlined such case in Example 7. Considering the simple case of a binary signal with a T-energy pulse $g(t)$, the received signal is

$$r(t) = \sqrt{P_r} \sum_{n=-\infty}^{+\infty} a[n]g(t-nT) + w(t) \tag{2.89}$$

where $w(t)$ is AWGN with psd $N_0/2$ and P_r is the received signal power. The received energy per data symbol E_s is also $E_s = P_r T$. The detection problem is easily formulated: the receiver has to recover (or *regenerate*) the transmitted data stream upon observation of $r(t)$. Unfortunately, the presence of the random noise $w(t)$ hinders such function. When the particular *realization* of the random process is particularly unfavorable, it may happen that the regenerated datum $\hat{a}[n]$ is different from $a[n]$, and a *symbol* error is produced.

"When" and "where" an error is produced is not predictable in advance, due to the random nature of noise and data. What we can evaluate upon knowledge of the statistical properties of the noise (and of the data as well) is the *probability* that a symbol error (or bit error) occurs. So the preeminent performance index of the communication link is the *Symbol Error Rate* (SER) or the *Bit Error Rate* (BER), where *rate* is the practical interpretation of *probability*. The detection problem is thus formulated as follows: *find the signal processing that we ought to apply to $r(t)$ to regenerate the data symbol stream $a[n]$ with the minimum SER*.

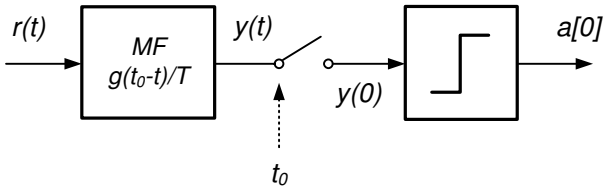
This issue is not actually simple to solve, so let us tackle it step by step. Assume that we have to detect a *single* isolated datum $a[0]$ with its associated pulse $g(t)$, i.e., $r(t) = \sqrt{P_r}a[n]g(t) + w(t)$. This problem was solved back in the 50's with an elegant geometric interpretation based on expansion of $r(t)$ onto a Hilbert space of orthonormal functions. All in all, such solution is equivalent to the processing scheme shown in Fig. 2.23 (a) for PAM signals with equiprobable i.i.d. symbols, and is called the *matched filter receiver*. As was anticipated in Example 2.51, the signal is processed by an LTI filter matched to $g(t)$, i.e., with an impulse response[1] $h_{MF}(t) = g(t_0 - t)/E_g$, with an appropriate t_0. The resulting signal is

$$y(t) = \sqrt{P_r}a[0]g_r(t) + n(t) \tag{2.90}$$

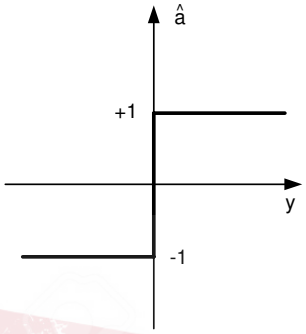
where $g_r(t) = g(t) \otimes h_{MF}(t)$ is the filtered data pulse, and $n(t)$ is filtered zero-mean Gaussian noise with psd function $S_n(f) = N_0|H_{MF}(f)|^2/2$. The filtered signal is evaluated at the time instant t_0 to get the following "soft" (i.e., real-valued) sample

$$y(0) = \sqrt{P_r}a[0]g_r(t_0) + n(0) \tag{2.91}$$

[1]The impulse response cited in the text is optimum for real-valued pulses $g(t)$. If the pulse is complex-valued, the impulse response of the matched filter is $h_{MF}(t) = g^*(t_0 - t)/E_g$



(a)



(b)

Figure 2.23 Matched filter receiver for baseband data signals (a) and nonlinear element characteristic (b)

For an NRZ pulse, $t_0 = T$, whereas for a symmetric SRFRC pulse, $t_0 = 0$. In both cases, $g_r(t_0) = 1$. Also, $n(0)$ is a Gaussian random variable with zero mean and variance

$$\sigma_n^2 = \frac{N_0}{2}\int_{-\infty}^{+\infty}|H_{MF}(f)|^2 df = \frac{N_0}{2E_g^2}\int_{-\infty}^{+\infty}|G(f)|^2 df = \frac{N_0}{2E_g} = \frac{N_0}{2T} \tag{2.92}$$

The soft output $y(0)$ (also called *decision variable*) is finally passed through a "hard detector" that implements the threshold nonlinearity shown in Fig. 2.23 (b) to change the continuous-amplitude sample $y(0)$ into the regenerated discrete-value symbol $\hat{a}[0]$. The probability of a symbol error (that, in our case of binary signaling, is equal to that of a bit error) is

$$\begin{aligned} SER = BER &= \frac{1}{2}\Pr\{y(0) > 0 \mid a[0] = -1\} + \frac{1}{2}\Pr\{y(0) \le 0 \mid a[0] = +1\} \\ &= \frac{1}{2}\Pr\{-\sqrt{P_r} + n(0) \le 0\} + \frac{1}{2}\Pr\{\sqrt{P_r} + n(0) \le 0\} \\ &= \Pr\{n(0) > \sqrt{P_r}\} = Q\left(\frac{\sqrt{P_r}}{\sigma_n}\right) = Q\left(\sqrt{\frac{2E_s}{N_0}}\right) \end{aligned} \tag{2.93}$$

where $Q(\cdot)$ is the familiar Gaussian integral function

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}}\int_{-\alpha}^{+\infty}\exp(-\beta^2/2)d\beta \tag{2.94}$$

With a computation only slightly more complex, it is easily found that the SER for generic M-PAM with alphabet (2.80) is

$$SER = 2\left(1 - \frac{1}{M}\right)Q\left(\sqrt{\frac{3}{M^2-1}}\sqrt{\frac{2E_s}{N_0}}\right) \quad \text{(M-PAM)} \tag{2.95}$$

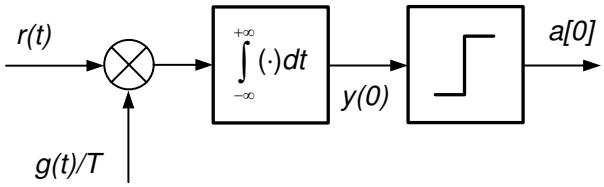


Figure 2.24 Correlation receiver for baseband data signals

Instead of the ratio E_s/N_0 it is sometimes more expedient to express the SER/BER as a function of the ration between the received energy *per bit* E_b and the noise psd N_0. For M-PAM, $E_s = \log_2(M) \cdot E_b$ and so

$$SER = 2\left(1 - \frac{1}{M}\right) Q\left(\sqrt{\frac{3\log_2(M)}{M^2-1}}\sqrt{\frac{2E_b}{N_0}}\right) \quad \text{(M-PAM)} \tag{2.96}$$

A variant of the matched filter receiver that is equally optimum on the AWGN channel is depicted in Fig. 2.24. Take back into consideration (2.90) that describes the output of the matched filter for one-shot data transmission. We have

$$y(t) = r(t) \otimes h_{MF}(t) + n(t) = \int_{-\infty}^{\infty} r(\alpha) h_{MF}(t-\alpha) d\alpha$$

$$= \frac{1}{E_g}\int_{-\infty}^{\infty} r(\alpha) g(t_0 + \alpha - t) d\alpha \tag{2.97}$$

The filtered received signal is evaluated at $t = T_0$, so that the soft decision variable is ($E_g = T$)

$$y(t_0) = \frac{1}{T}\int_{-\infty}^{\infty} r(\alpha) g(\alpha) d\alpha = \frac{1}{T}\int_{-\infty}^{\infty} r(t) g(t) dt \tag{2.98}$$

This is just what is depicted in Fig. 2.24. The operation that is performed is the *correlation* between the received signal $r(t)$ and a local waveform that corresponds to transmission of a data bit equal to 1. The correlation receiver is just an alternative arrangement of the matched filter receiver, and so its BER performance is exactly the same.

The same results concerning the BER/SER of the matched filter/correlation receiver can be arrived at if we *normalize* equation (2.91). If we divide both the signal and the noise term (thus leaving the SNR unchanged) by the standard deviation $\sigma_n = \sqrt{N_0/2T}$ of $n(0)$ we get

$$y(0) = \sqrt{\frac{2E_s}{N_0}} a[0] g_r(t_0) + n_1(0) \tag{2.99}$$

where $n_1(0)$ is a zero-mean Gaussian random variable with *unit* variance. We may alternatively normalize by $\sqrt{P_r}$ obtaining

$$y(0) = a[0] g_r(t_0) + N(0) \tag{2.100}$$

where now $N(0)$ is a zero-mean Gaussian random variable with variance $\sigma_N^2 = N_0/(2E_s)$. We explicitly mention these alternative formulations since in the following we will use indifferently one of the three expressions (un-normalized (2.91), unit-variance (2.99), unit-amplitude (2.100)) that we introduced here.

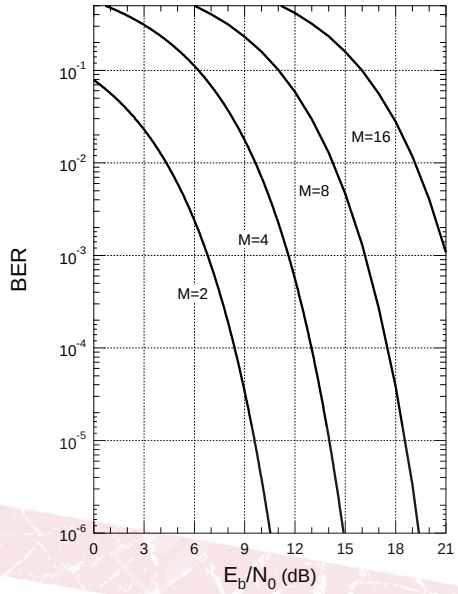


Figure 2.25 BER of the matched filter receiver for M-ary PAM with no ISI

Expression (2.93) for the BER/SER is called the *matched filter bound* and its appearance as a function of the E_b/N_0 ratio (that is basically a measure of the SNR experienced on every data symbol) is shown in Fig. 2.25. Why is this called a *bound*? Because this is what we would get for *one-shot* transmission of a *single* data symbol.Taking back into consideration the adjacent symbols, what we actually get at the output of the matched filter is

$$y(t) = \sqrt{P_r} \sum_{n=-\infty}^{+\infty} a[n] g_r(t - nT) + n(t) \tag{2.101}$$

that, evaluated at t_0 gives

$$\begin{aligned} y(t_0) &= \sqrt{P_r} \sum_{n=-\infty}^{+\infty} a[n] g_r(t_0 - nT) + n(t_0) \\ &= \sqrt{P_r}\, a[0] + \sum_{n \neq 0} a[n] g_r(t_0 - nT) + n(t_0) \end{aligned} \tag{2.102}$$

In addition to the data symbol to be detected and to the noise term as in the "one-shot" computation above, we also have an additional term (the intermediate one) that is due to the presence of trailing and leading symbols in the data stream, and is perceived as an interference of them on $a[0]$: the *InterSymbol Interference* (ISI). The ISI makes the BER degrade with respect to the value (2.93) given by the matched filter bound. Of course, not any pulse shape $g_r(t)$ arises ISI. If we have

$$g_r(t_0 - nT) \equiv 0\ ,\ n \neq 0 \tag{2.103}$$

then the ISI is no longer present, and we experience again the BER of the matched filter bound. The condition (2.103) is for instance actually satisfied by a (symmetric) NRZ pulse

filtered by its matched filter. The resulting $g_r(t)$ is a triangular pulse as in Fig. 2.20 (b), and it is seen that its symbol-rate samples are all zero apart from the one at $t = t_0 = 0$. The same is true for the SRFRRC pulse, although is may not seem so at a first glance. To see this, take into consideration that the impulse response of the matched filter to an SRFRC is basically another SRFRC pulse, since $g(t)$ is even-symmetric. If we consider FTs, we have

$$G_r(f) = G(f) \cdot H_{MF}(f) = G(f) \cdot G^*(f)/E_g$$

$$= \sqrt{TG_N(f)} \cdot (1/T)\sqrt{TG_N(f)}/T = G_N(f) \tag{2.104}$$

and so $g_r(t) = g_N(t)$. From (2.14) and from Fig. 2.7 it is easily seen that all samples of the Nyquist pulse $g_N(t)$ at $t_n = nT$ are null, apart from the one with $n = 0$ that is equal to 1. The same conclusion can be actually drawn with frequency-domain considerations. Take in fact again (2.103) with $t_0 = 0$. The sequence of symbol rate samples of $g_r(t)$ is such that $g_r(nT) = \delta[n]$. Therefore, taking the FT of both sequences and considering Poisson's relation (2.70)), we have

$$R \sum_{k=-\infty}^{+\infty} G_r\left(f - kR\right) = 1 \tag{2.105}$$

that is called the *Nyquist criterion* for the absence of ISI. If we replace G_r with G_N as above, we easily see that (2.105) is satisfied by virtue of the particular symmetry around $f = R/2$ of the roll-off region of $G_N(f)$ in Fig. 2.7, for any value of β.

We can now motivate the adoption of the SRFRC pulse in almost all bandlimited data communication systems currently in use: such pulse give rise to no ISI when matched-filter detected, combining optimal immunity against noise with the absence of ISI on a bandlimited channel.

2.7 Modulation, Demodulation and Modem Architecture

2.7.1 Linear I/Q digital modulation

It is widely known that a baseband signal like those encountered in the previous section is not suited for transmission over a wireless channel. Radio signals can be efficiently detected only when the size of the transmitting/receiving antenna is of the order of magnitude of the wavelength of the electromagnetic wave. Just to make an example, the wavelength of wave oscillating at a frequency f_0=100 kHz is $\lambda_0 = c/f_0$=3 km, thus making transmission of such a component really problematic.The solution to this is using a high-frequency carrier (for instance, in Wi-Fi wireless LANs the carrier frequency is f_0=2.4 GHz) and attaching to that carrier the digital data to be sent by *modulating* the amplitude and/or the phase of the carrier with the data bearing baseband signal much like what described in Sect. 2.4 concerning the general scheme of an I/Q modulator of Fig. 2.15.

When we intend to perform a digital modulation, what we have to specify in that general scheme is how the I/Q baseband components of the bandpass signal are related to the digital data to be sent. In a sense, the different modulation schemes are different, specialized "front-ends" to the general I/Q modulator.

The simplest digital modulation is BPSK (Binary Phase-Shift Keying) wherein the Q component $x_Q(t)$ is null, and the I component is just an NRZ binary signal like (2.79), so

that the resulting complex envelope is

$$x(t) = A \sum_{n=-\infty}^{+\infty} a[n]g(t-nT) + \jmath 0 \;\; \text{(BPSK)} \tag{2.106}$$

At each time instant, the In-phase carrier $\cos(2\pi f_0 t)$ is either multiplied by a positive value, or by a negative one, according to the polarity of the data to be transmitted. Therefore, data modulation amounts to either not changing the phase of the carrier, or by *shifting* it by 180 degrees.

2.7.2 Signal constellations

Many variants of digital modulation exist. We stick here for simplicity to *linear* schemes, wherein the I and Q baseband components $x_I(t)$ and $x_Q(t)$, hence the complex baseband equivalent $x(t)$ of the modulated signal $x_{BP}(t)$, is obtained from the data bits with *linear* operations only. The very form of the BPSK signal (2.106) can be seen as a linear interpolation with pulse $g(t)$ of the digital stream $a[n]$. The simpler advancement to BPSK involves the use of the Q component that in BPSK is absent. To do so, we build an NRZ signal with a decimated version of the binary stream containing only even-index data $a[2m]$, and another NRZ signal with the odd-numbered bits $a[2m+1]$. Using the two signals as x_I and x_Q respectively, gives a *Quadrature Phase-Shift Keying* signal

$$\begin{aligned} x(t) &= A \sum_{m=-\infty}^{+\infty} a[2m]g(t-mT) + \jmath A \sum_{m=-\infty}^{+\infty} a[2m+1]g(t-mT) \\ &= A \sum_{m=-\infty}^{+\infty} \left(a[2m] + \jmath a[2m+1]\right) g(t-mT) \;\; \text{(QPSK)} \end{aligned} \tag{2.107}$$

At each time instant, the signal $x(t)$ takes one of four possible values given by $A(\pm 1 \pm \jmath)$ that lie on a circle in the complex plane with radius $\sqrt{2}A$, as represented in Fig. 2.26 (b). The four points have constant amplitude but four different phases, hence the denomination of quadri-phase signal. A BPSK signal on the contrary is characterized by two points only, both lying on the real axis as in Fig. 2.26 (a). It is apparent that for the same signaling rate $R = 1/T$, the two signals achieve different bit rate. BPSK has $R_b = R$, whilst with QPSK $R_b = 2R$, akin to what happens with a four-level signal.

The appearance of the QPSK signal in (2.107) suggests the general form for the "front-end" of a linear data modulator that is shown in Fig. 2.27. The input binary data stream $a[n]$ with a bit-rate R_b is segmented into *words* of N_b bits each, with a word rate $R_w = R_b/N_b$. Every word is used to identify a specific *symbol* in the complex plane selected in a set (the *alphabet*) of $M = 2^{N_b}$ elements. The representation of the symbols in the set is also called the *constellation* of the digital modulation. From the previous example, for QPSK we have $N_b = 2$ and $M = 4$. Higher-order M-PSK constellations with M equal to 8, 16 or 32 are characterized by M points evenly distributed on a circle in the complex plane. We show in Fig. 2.28 (a) an 8-PSK constellation where each symbol is labeled by the particular pattern of 3 bits (the value of the 3-bit word) that causes its own transmission. The correspondence between bit words and symbols in the constellation is the *mapping* that is implemented by the relevant block in Fig. 2.27. Upon generation of N_b bits, the mapper generates at time mT a new complex-valued symbol $s[m] = s_I[m] + \jmath s_Q[m]$ selected in the constellation. The symbol rate is $R = 1/T = R_b/N_b$.

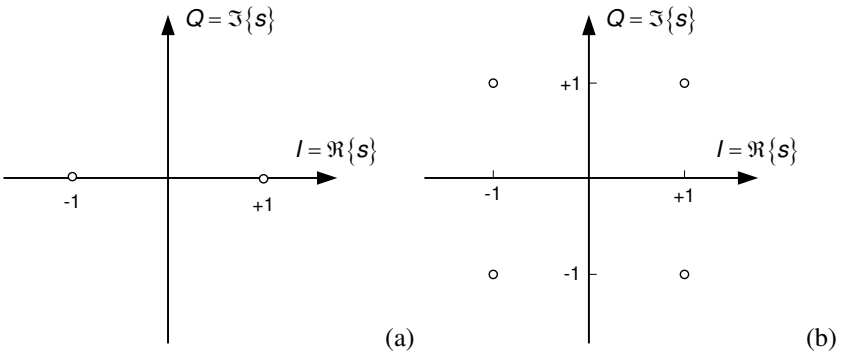


Figure 2.26 Representation of BPSK (a) and QPSK (b) signals on the complex plane ($A = 1$)

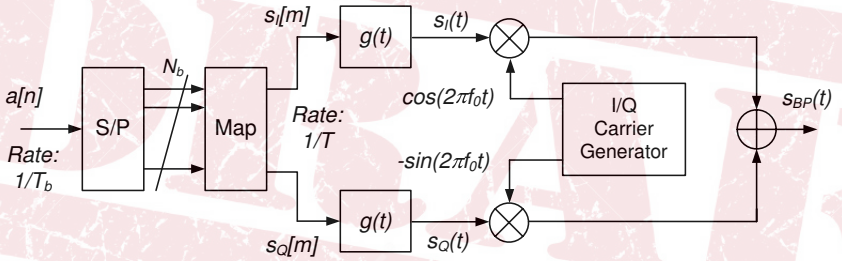


Figure 2.27 Linear digital data modulator

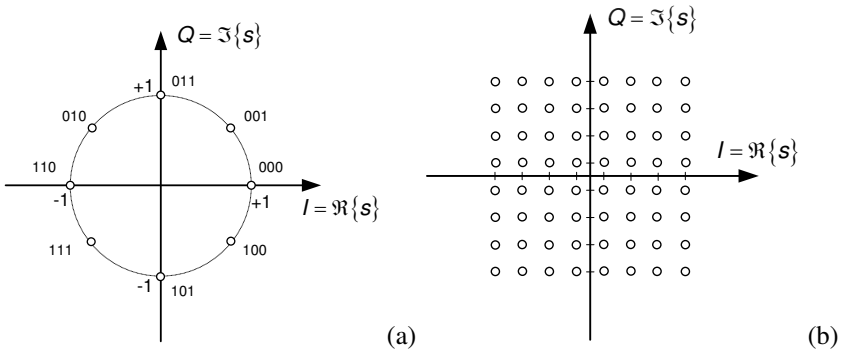


Figure 2.28 8PSK (a) and 64-QAM (b) constellations - ($A = 1$)

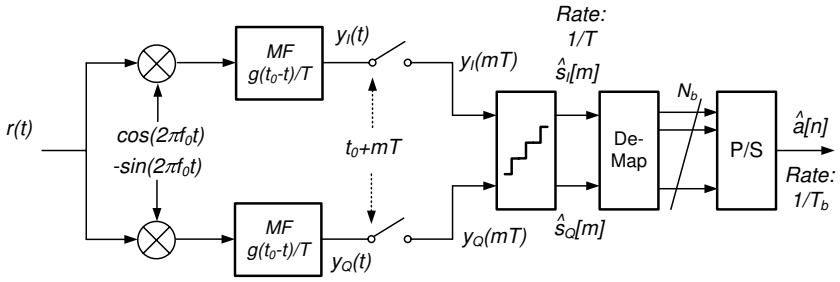


Figure 2.29 I/Q Digital data demodulator

As in baseband transmission, modulated signals need some form of bandlimiting to fit a particular bandwidth (centered on the carrier frequency f_0) that is assigned by some regulatory body. Such function is accomplished by bandlimiting the baseband equivalent of the signal prior to I/Q modulation. This is why the front-end in Fig. 2.27 features a shaping filter (interpolator) with (bandlimited) pulse $g(t)$, usually a SRFRC function. The general form of the signal produced by the modulator is

$$s(t) = A \sum_{m=-\infty}^{+\infty} s[m] g(t - mT) \tag{2.108}$$

where $s[m]$ is the m-th complex symbol taken from a constellation to be specified, for instance the square 64-QAM constellation in Fig. 2.28 (b), that is produced by the composition of two 8-level signals with alphabet $\{-7, -5, -3, -1, +1, +3, +5, +7\}$ on each component.

2.7.3 Demodulation of Linear I/Q modulated signals

Demodulation of a digital signal is straightforward if we appropriately combine (we would say "cascade") the I/Q demodulator as in Fig. 2.16 with matched filter detection (Fig. 2.23). In particular, the I/Q demodulator has to be followed by a "back-end" that reverse the operation of the front-end in Fig. 2.27, resulting in the schematic shown in Fig. **??**. The low-pass filter of the I-Q demodulator in Fig. 2.16 no longer appears since the double-frequency filtering function is performed by the matched filters. At the output of the signal samplers we get two "soft" values that represent a noise-corrupted version of the I/Q components of the transmitted symbol $s[m]$. If we collect a number of such samples and we report such values as dots on the complex plane, what we get is called a *scatter plot* and appears as in Fig. 2.30 for a 16-QAM constellation.

The "complex slicer" in Fig. 2.16 is the complex counterpart of the threshold detector for baseband signals. The complex plane is partitioned into different zones, the so-called *decision regions* according to a decision rule, so that digital symbols are regenerated from the soft outputs $y(mT)$. Each region is labeled by a constellation symbol so that whenever $y(0)$ lies in a certain region, the "label" symbol is regenerated. The regenerated symbols $\hat{s}[m]$ are finally de-mapped to get back the N_b-bit word ($N_b = 4$ in the example) corresponding to the regenerated symbol. The function of the cascade of complex slicer and demapper is represented in Fig. 2.31, where we directly indicate the estimated N_b-bit word after demapping. The labels of the decision regions are directly the demapped bits.

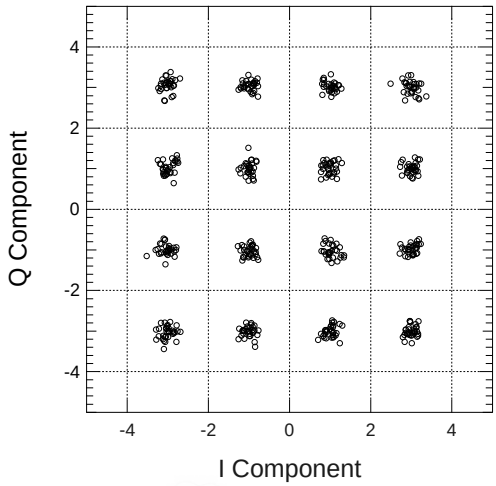


Figure 2.30 Scatter plot of noisy 16QAM samples at the matched filter output

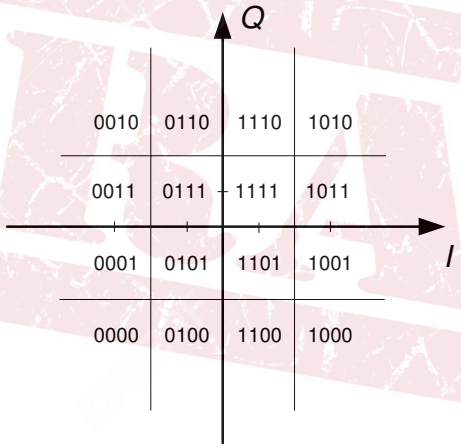


Figure 2.31 Decision regions for a 16QAM signal with demapping

When the noise component is large, $y(mT)$ may cross the boundaries of the decision region pertaining to $s[m]$, and a symbol error is produced: $\hat{s}[m] \neq s[m]$. The computation of the SER for a generic complex constellation is simple but tedious, so that we will not dwell here on such issue. A very tight bound, especially when the SER is small, is

$$SER \leq 4\left(1-\frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3}{2(M-1)}}\sqrt{\frac{2E_s}{N_0}}\right) \quad \text{(M-QAM)} \tag{2.109}$$

2.7.4 Architecture of DSP-based Data Demodulators

How are the techniques for signal modulation and demodulation that were described in the previous Sections actually implemented in practical realizations? The current trend

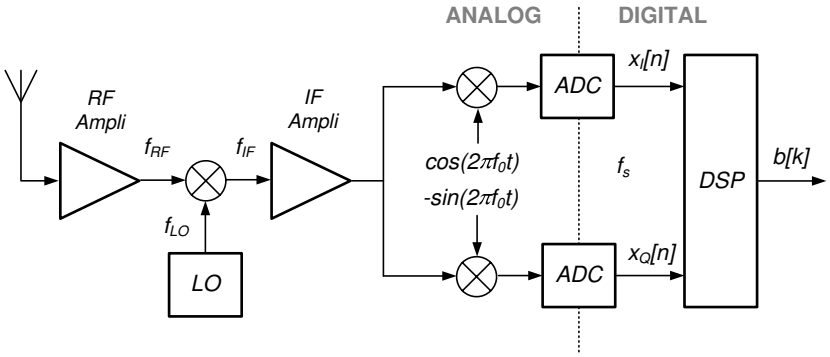


Figure 2.32 I/Q Signal Demodulation with BaseBand sampling

in modem design, both in a cheap Mobile Terminal (MT) of a cellular network (UMTS, GSM) and/or in an expensive Radio Base Station, is to perform as much signal processing in the digital domain as possible (modulation, demodulation, filtering etc.). In the receiver section of the modem (the most challenging to implement) we have basically two possible alternatives as far as the location of the ADC stage is concerned. The first approach, sketched in Fig. 2.32, is mainly pursued in low-power MTs. The RF received signal is converted to base-band using a conventional I/Q analog tuner followed by twin I/Q ADCs with sampling frequency f_s, and base-band digital signal processing for data detection. In the lowest-cost, lowest-power implementations, the first conversion stage to IF is absent, and the RF signal is directly brought to (I/Q) baseband (so-called *zero-IF* architecture).

The critical points of this arrangement are the possible amplitude imbalance of the two I and Q analog rails, as well as the imperfect quadrature between the two I/Q carriers used for IF to baseband conversion. A precision receiver with loose consumption or cost constraint is implemented according to the IF-sampling architecture shown in Fig. 2.33 (a), where the ADC stage is shifted towards the antenna. Incidentally, we observe that this is just the general scheme of the so-called *software-defined radios* (SDR), where all of the signal processing, apart from the initial RF-to-IF conversion is performed in the digital domain, and the DSP components are reprogrammable to a certain extent. With an SDR, changing the signal format to be handled just amounts to changing the software that drives the (programmable) DSP components, instead of changing a piece of dedicated hardware (a card, or the whole modem) as in conventional equipment.

Coming back to Fig. 2.33 (a), the ADC operates directly on the IF signal, and the relevant digital output is still a bandpass signal on a different, digital, intermediate frequency $f_{DIF} = f_{IF} \pm k \cdot f_{sa}$ (k an integer). The analog front-end is simplified, and the task of base-band conversion is deferred to the digital section (with no issues of amplitude imbalance and imperfect quadrature). The main drawback of the IF-sampling approach is the tighter requirements for the ADC which must now handle faster, IF signals (instead of base-band signals as in Fig. 2.32). In particular, the converter rise and fall times, i.e., the time needed to ”open” and ”close” the gate of the sampling device shall be commensurate to the analog IF frequency, and turns out to be much shorter than those of the converters operating on the base-band signal. As a consequence, the IF-sampling ADC is in general more expensive and power-consuming than the two baseband converters in Figure 2.32. The “ultimate” version of a software-defined radio is shown in Fig. 2.33 (b), and is called

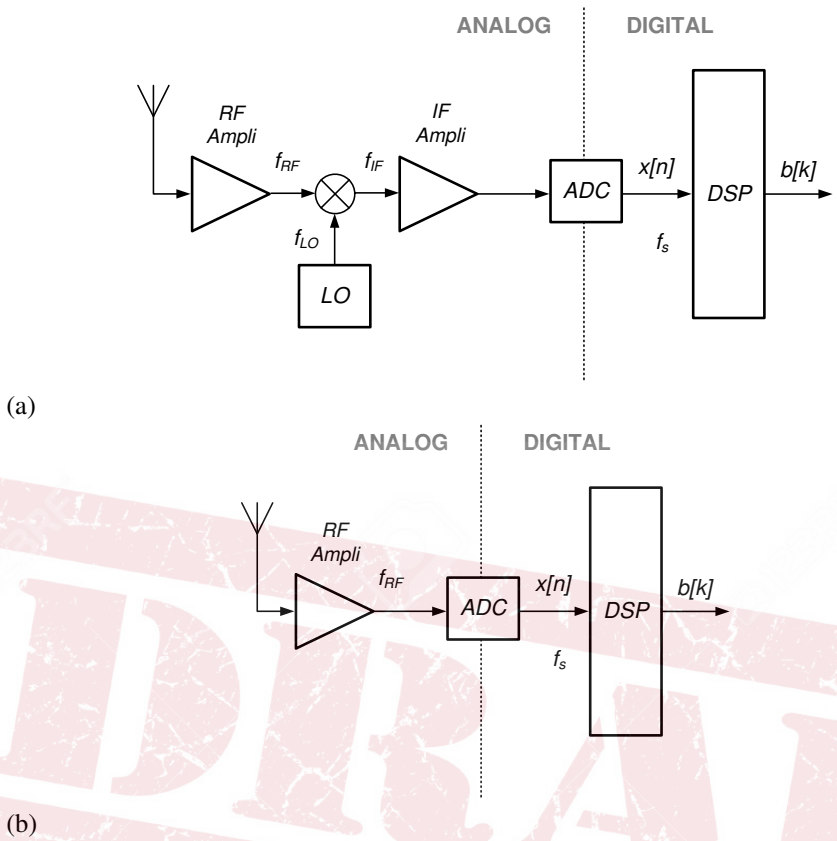


Figure 2.33 I/Q Signal Demodulation with IF sampling (a) or with RF sampling (b)

a *direct-RF-conversion* receiver. The ADC is further moved towards the antenna, and the analog components are reduced to the bare minimum, in particular no IF conversion is needed. Of course, the issues related to the ADC cost and power consumption that were already mentioned regarding the IF-sampling architecture are here further exacerbated. Nonetheless, this kind of architecture is used in military equipment where special needs such as fast signal acquisition and detection, as well as total terminal reconfigurability, are of paramount relevance.

The notion of an SDR that was presented here for digital receivers has to be intended as also applicable to the corresponding digital transmitters, where the general schemes that we showed in Fig. 2.33 (a) are to be "reversed", with the due minor modifications that we leave to the reader to discover.

CHAPTER 3

REDUNDANCY AND EFFICIENCY - DATA COMPRESSION

“ . . . – – – . . . ”

—Table of the celebrated Morse code for telegraphy, invented by Samuel Morse (1791-1862) in 1837

The Morse code is undoubtedly the first widely used example of digital source coding, with variable codeword lengths to minimize the average message length.

3.1 What is "Source Coding" ?

Within the Internet, all signals to be transmitted are represented digitally. Through appropriate coding operations, audio, video, voice, or image signals are "translated" into a stream of binary symbols (bits) that are either to be transferred over a communication link, or stored and reused later. This encoding operation must be performed optimally, so that the data flow generated in terms of bits/s is not too burdensome for the network supporting communication and/or storage. The communication speed (bit rate) R_b in bits/s represents the capacity (sometimes incorrectly called bandwidth) that the network must make available to the source. The greater the number of bits produced by the encoding operation, the greater the required capacity (the size of the resulting file) and therefore the cost of the service.

Example 3.10

An HDTV digital video signal consists of a stream of 25 (in Europe) frames (i.e., images) per second. Each image is composed of 1920 (on the horizontal axis H) times 1080 (on the vertical axis V) picture elements (i.e., pixels), each of which is characterized by 256 levels (8 bits) of the three primary colors Red, Green, and Blue. The total bit rate of the video stream in this format is equal to

$$R_{b,HDTV} = 25 \times 1920 \times 1080 \times 3 \times 8 = 1.24416 \text{ Gbit/s} \tag{3.1}$$

For the UHD (Ultra-High-Definition) or 4K format, we further multiply this value by 4 because the H/V resolutions are both increased by a factor 2, and we also wish to include the High Dynamic Range (HDR) format, in which the colors are represented on 10 bits instead of 8, we have a further increase of a factor of 10/8, and we get to

$$R_{b,UHD-HDR} = 25 \times 3840 \times 2160 \times 3 \times 10 = 6.2208 \text{ Gbit/s} \tag{3.2}$$

that is, a truly remarkable value and beyond the reasonable capacity of any (reasonable) digital connection today (2025).

Reducing the number of bits produced by the encoding process, that is, *compressing the source*, means increasing the overall throughput of the network and/or reducing the cost of transmitting a single stream. Or, if the information produced through encoding is to be stored (on a hard disk, on an SD card, etc.), reducing the information rate is equivalent to a virtual increase in the storage medium's capacity. This is the *raison d'être* of digital encoding and compression techniques for audio signals (MP3), voice signals for telephony (vocoders), video signals (MPEG, HEVC), still images (GIF, JPEG), as well as simple generic file compressors/expanders such as 7-zip, etc.

Before examining the theoretical foundations of such techniques in the following paragraphs, we will have a look at some basic examples of source compression. We will later be able to place them in a broader context, but they serve now as a starting point for our discussion.

3.1.1 Unbalanced probabilities

Sometimes, the binary symbols "1" and "0" produced by a digital source appear with very different frequencies. Consider scanning a text document to be saved in .pdf format (Fig.

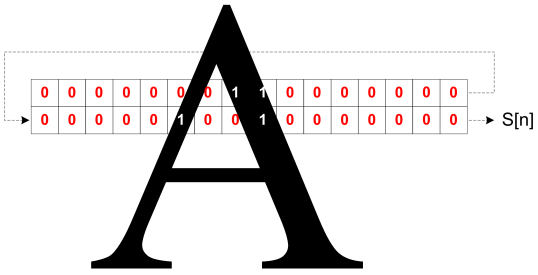


Figure 3.1 B/W scan of a text document

3.1): most of the pixels in the image are white and are encoded as "0", and only occasionally do you encounter black pixels encoded as "1". A snippet of the binary stream produced by the scanner may be as follows:

100000000100000000010000011000000100100000000000001000000010001000000010000001

We see that the "0" symbols tend to clump into long repetitions (runs) interspersed with a few "1" symbols. Before transmitting this stream, we then adopt a special encoding procedure: each sequence of "0"s enclosed between two non-consecutive "1"s is replaced by the initial "0" only, followed by a certain number of bits, say 3, which, in natural binary encoding, represent the total length of that particular "0" run. If the run is longer than 7 (maximum integer representable on 3 bits), the 0 is repeated and another 3 bits are inserted with the length of the remaining run, and so on. The given string becomes as follows:

101111011100011010111011010001101111011010111100101011110 1101

which is considerably shorter than the original string: the binary source has been *compressed* with a lossless compression algorithm. *Lossless* because we see that from the encoded string we can reconstruct the original string *exactly*, without loss of fidelity and therefore of information. Run length encoding (with slightly different methods than those presented here, but with the same operating principle) is embedded in many data compression algorithms as of today, including for instance JPEG encoding for images.

Example 3.11

The old stereo-CD format for high quality audio signals was based on a standard sampling frequency $f_s = 44.1kHz$ (adequate for a nominal audio bandwidth of 20 kHz) for the two stereo channels. Each sample is represented on a 16-bit digital word for best quality, leading to a total digital bit rate

$$R_{CD} = 44.1 \times 2 \times 16 = 1,411.2 \text{ kbit/s} = 1.4112 \text{ Mbit/s} \tag{3.3}$$

Such a bit rate corresponds to an unprocessed, or *raw* format that is stored on the CD without any further processing. The corresponding file format is called `.wav`.

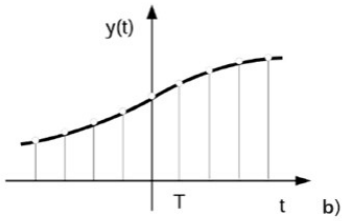


Figure 3.2 Sampling of a bandlimited signal

3.1.2 Lossy source coding

We have seen that CD-standard encoding of audio signals requires approximately 700 kbit/s per channel. Such a bit-rate is way too much for encoding of simple voice signals with telephone-grade quality. We can easily decrease the bit rate if some loss of fidelity, i.e., signal distortion, is accepted. In other word, the audio source can be significantly compressed using a *lossy* coding strategy that does not fully preserve the signal's information content, but only safeguards what is considered essential, as is done in good-old PCM encoding. In this strategy, the analog signal is first bandlimited to 4 kHz (the primary source of information loss), then sampled at 8 ksamples/s and finally quantized on 8 bits (rough quantization, the second source of fidelity loss). The resulting digital signal has a data rate of $R_{PCM} = 64$ kbit/s, less than 1/10 of the original data rate of CD-quality PCM. In general, by accepting varying degrees of information loss, very high source compression ratios (defined as the ratio between the data rate of the original and the compressed stream) can be achieved.

3.1.3 Correlated Information Sources

Let's now take a quantum leap, completely forgetting about the beautiful CD-quality signal at 700 kbit/s, and focusing instead on the PCM-encoded voice signal at 64 kbit/s as above. Is it possible to further reduce this rate without an appreciable further loss of fidelity? From this perspective, it is not advisable to go below 8 bits/sample. So, what feature should we exploit to further compress our source? Fig. 3.2 represents a few samples of our bandlimited voice signal. We see that the value of two consecutive samples is usually not very different, as if the next sample were somehow affected by the value of the previous one. This indicates a certain amount of *memory* in the signal source, in the sense that the current value of a sample is strongly *correlated* with a certain number of previous samples, and also indicates a practical way to further compress the source. Fig. 3.3 shows the *differential PCM* encoding scheme (DPCM). Quantization does not occur directly on the sequence $x[n]$, but on the sequence $e[n]$ obtained as the difference between $x[n]$ and a sequence $\hat{x}[n]$ whose n-th sample is obtained as a *prediction* of $x[n]$ based on the previous samples of a certain version of $x[n]$, called $x_Q[n]$, affected by quantization error. To understand this, observe that the *prediction error* $e[n] = x[n] - \hat{x}[n]$; $e[n]$ is quantized, obtaining $e_Q[n]$ so that, as is apparent,

$$x_Q[n] = \hat{x}[n] + e_Q[n] = x[n] - e[n] + e_Q[n] = x[n] - q[n] \quad , \quad q[n] \stackrel{\triangle}{=} e[n] - e_Q[n]$$

where $q[n]$ is the *quantization error*.

As will become clear shortly, the signal $x_Q[n]$ is the *reconstruction* of $x[n]$ that a DPCM *decoder* is able to produce. The prediction guarantees good results by virtue of the high

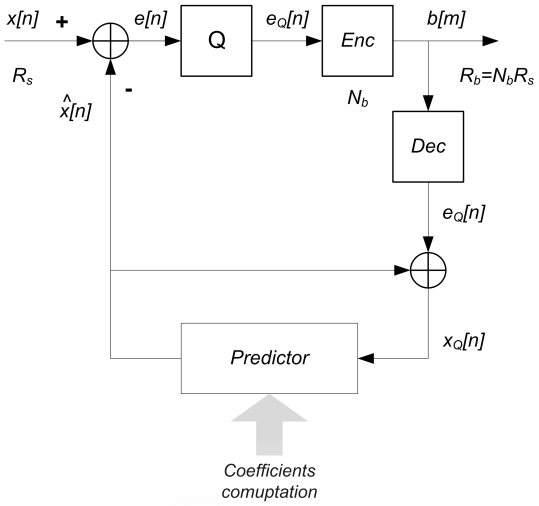


Figure 3.3 (A)DPCM Voice encoder

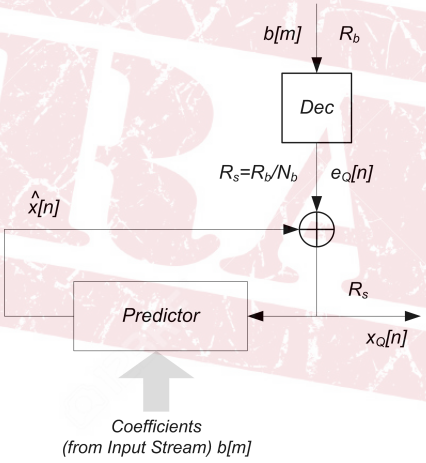


Figure 3.4 (A)DPCM Decoder

correlation between consecutive samples of $x[n]$, and therefore the prediction error signal $e[n]$ is a "small" signal on average (i.e., with a smaller rms value than that of $x[n]$) and can therefore be quantized onto a smaller number of bits, attaining higher compression ratios with no quality degradation.

To get an idea about how a signal value can be predicted, and about how the correlation between adjacent samples provides the possibility of signal compression, we study the *one-step optimal linear predictor*. In this case, the predicted signal $\hat{x}[n]$ is obtained as a linear combination of the immediately preceding *reconstructed* signal sample:

$$\hat{x}[n] = ax_Q[n-1] = a(x[n-1] - q[n-1]) \tag{3.4}$$

where the value a is such that the mean squared prediction error is minimum:

$$\varepsilon \stackrel{\Delta}{=} E\{|\hat{x}[n] - x[n]|^2\} = \min \tag{3.5}$$

Elaborating, we have

$$\begin{aligned}\varepsilon &= E\{|ax[n-1] - aq[n-1] - x[n]|^2\} \\ &= E\{a^2x^2[n-1] + a^2q^2[n-1] + x^2[n] - 2ax[n]x[n-1]\} \\ &= a^2(R_x[0] + \sigma_q^2) - 2aR_x[1] + R_x[0] \end{aligned} \tag{3.6}$$

where we have assumed that $x[n]$ is a stationary process, and we have indicated with $R_x[m] \triangleq E\{x[n]x[n-m]\}$ the autocorrelation sequence of such process, and where we σ_q^2 is the power of the quantization error, that we also assume being independent of the signal value. (3.6) is a quadratic function in a that is minimized for the following optimal value:

$$a_{OPT} = R_x[1]/(R_x[0] + \sigma_q^2) \tag{3.7}$$

If the signal samples were uncorrelated, $a = 0$, the signal would be unpredictable, and DPCM coding would be useless. In general, the predictor is a causal function of N previous signal samples, $\hat{x}[n] = \mathcal{F}(x[n-1], x[n-2], \ldots, x[n-N])$, very often implemented as a linear FIR filter

$$\hat{x}[n] = \sum_{k=1}^{N} a_k x[n-k]$$

whose coefficients are to be optimized as above.

DPCM decoding is depicted in Fig. 3.4: the prediction filter reconstructs the current value of $\hat{x}[n]$ from the previous samples of $x_Q[n]$ right as in the encoder, and the predicted value is then added to the quantized prediction error $e_Q[n]$ input to the decoder itself, extracted from the received stream, to get the output signal $x_Q[n]$. It is seen that the encoder essentially contains a replica of the decoder to reconstruct the decoded signal $x_Q[n]$ from which the prediction is calculated.

The reason why the encoder's predictor operates on $x_Q[n]$ samples rather on $x[n]$ is that the encoder and the decoder must provide the very same prediction. If the encoder used $x[n]$, the predicted values $\hat{x}[n]$ would be different from those computed by the decoder, that does not have $x[n]$ - to avoid this misalignment, the encoder is also forced to use something also available to the decoder, namely, $x_Q[n]$.

In the particular case of speech, it is necessary to make the predictor *adaptive*, because such signal is not stationary: vowels are to a good approximation segments of a periodic, very well predictable signal, whilst consonants such as "s" or "f re much more more noise-like. To cope with such changes, the coefficients of the predictor filter are adapted at a certain rate to the variable conditions of the speech signal, typically every 10 ms. In such a way, prediction is always optimal and there is a substantial reduction in the number of bits needed to quantize $e[n]$ (4 bits instead of 8).

The coefficients of the predictor are computed real-time by the encoder and are multiplexed with the output stream so that the decoder can use them – such addition has a marginal impact on the overall data rate. ADPCM (Adaptive Differential PCM) coding has been standardized in the ITU G.721 standard for an ancoding rate of 32 kbit/s, thus achieving a compression factor 2 wrt standard PCM, and with a virtually identical subjective signal quality. This coding is used in DECT cordless phones for communication between the handset and the base.

The three examples of source coding that we have just introduced are based on different criteria (probability, fidelity, memory). They will be paradigmatic of the techniques that can be developed through the study and systematic application of *Shannon information theory*.

3.2 Lossless coding of a memoryless digital information source

We can characterize a digital information source as a device or a system (a video encoder, a digital voice recorder, a person texting via Whatsapp...) that produces a sequence of discrete random variables $S[n]$ (i.e., discrete-time random process) whose values, called symbols, belong to a certain finite sample space $\Omega = \{s_1, s_2..., s_M\}$ called the *alphabet*. The source can be binary like the B/W text scanner ($M = 2$), or multi-level, like the letters of the English alphabet in a file ($M = 26$). The various symbols are therefore the various values $S[0]$ $S[1]$ $S[2]$ ... taken by the variables in a particular realization of the random sequence. Neglecting the time dependence, we will identify the source with the name of the random variables of the sequence, namely S.

Speaking of real-time communications, the rate at which symbols of S are generated is measured in symbols/s. As already stated, the alphabet Ω could be binary $\{0, 1\}$ when the source is a generic data file, it could be the international alphabet $\{a, b, c, d, \ldots, x, y, z, \ \}$ including spaces if the source is text, $\{0, 1, \ldots, 255\}$ if the source is a grey-level image, and the symbol represents the brightness level of a pixel.

Generation of a symbol by a digital information source (hereinafter, briefly: source) corresponds to the occurrence of an event, specifically the event $\{S[n] = s_m\}$ (in short, and with an abuse of notation, event s_m) for some m, $1 \leq m \leq M$. When the symbols generated by S are statistically independent the source is *memoryless*. In this case the characterization of the source is complete when the probabilities of the diverse symbols n the alphabet is known:

$$p_m \triangleq \Pr\{S[n] = s_m\} \tag{3.8}$$

Here, we assume that such properties are invariant with n, i.e., the source is also stationary. When on the contrary $S[n]$ is statistically dependent on some $S[m]$, $m \neq n$, then the source has memory. As seen in the previous section, the speech encoder is a source with memory, while typically the bits flowing over an Internet connection can be considered a memoryless binary source.

We examine now a general procedure to encode a memoryless. First, we need to establish the *code alphabet*, onto which the source symbols will be remapped. Let's denote this new alphabet (sometimes called a dictionary) with $\Delta = \{d_1, d_2, ..., d_L\}$. In general, Δ is binary to make it easy to transmit or store the resulting stream. A code $\mathcal{M}$ will then be a correspondence (map) that associates groups of Ω symbols with groups of Δ symbols, called *codewords*. In particular, a block code maps each of the individual symbols s_m of Ω ($m = 1, 2, \ldots, M$) into a well-defined and fixed sequence of symbols of Δ, that is, into a certain codeword that we will call μ_m, as in the following example of a hypothetical Martian alphabet with only 4 letters:

- Source alphabet: $\Omega = \{a, b, c, d\}$
- Code alphabet: $\Delta = \{0, 1\}$
- Code table: $\mathcal{M}_1 : \begin{cases} s_1 = a \rightarrow \mu_1 = 0 \\ s_2 = b \rightarrow \mu_2 = 10 \\ s_3 = c \rightarrow \mu_3 = 11 \\ s_4 = d \rightarrow \mu_4 = 10 \end{cases}$

We see that this code is *variable-length* because the number of symbols in any codeword is not the same. ASCII encoding to represent alphanumeric characters is another example

of a *fixed-length* source encoding with a 127-symbol source alphabet and encoding with 7-symbol binary word (256 codewords with 8-bit encoding in the extended version). Using this code, it is possible to encode any string (i.e., a finite sequence) of source symbols to send them over an Internet link or store them in a file and later read them back.

In general, one wishes to be able to reconstruct (decode) the source string *exactly* (i.e., *losslessly*) from the encoded bits. From this perspective, the code (**??**) is useless because it is *singular*: the same block of code symbols (i.e., the same codeword) corresponds to *two* different source symbols, and it is therefore impossible to decode the code. The decoding operation is just the *reverse map* of the code, which allows us to trace the codewords back to the source blocks. A non-singular code could be:

$$\mathcal{M}_2 : \begin{cases} a \to 0 \\ b \to 10 \\ c \to 11 \\ d \to 00 \end{cases}$$

which, however, is not yet decodable. In fact, block 00 is ambiguous: it could refer to a sequence of two source letters a or to the single letter d. What we are looking for in source encoding is a *uniquely decodable* code, that is, one that does not create ambiguity in the reconstruction of the source message.

It's not easy to check whether a given code is uniquely decodable, and the example just given demonstrates this. Let us make an exhaustive list of the encoded string corresponding to any possible source string of length $K = 2$: We may consider this table as that of a new code, obtained by extension of the native code to blocks of symbols of the source of length $K = 2$ – it is not actually a new code, rather a trivial extension of the given code. We easily see that this new code is singular: the symbols ad and da are mapped to the same word 000. This indicates that problems may arise in decoding the code $\mathcal{M}_2$, as already mentioned. We conclude that a code is *uniquely decodable* if (and, it can be shown, only if) the generic K-th extension of the code itself is nonsingular for any (arbitrarily large...) K – a conceptually simple condition but very difficult to verify in practice.

Examine now two distinct, uniquely decodable codes for the usual quaternary Martian source:

$$\mathcal{M}_3 : \begin{cases} a \to 00 \\ b \to 01 \\ c \to 11 \\ d \to 10 \end{cases} \quad e \quad \mathcal{M}_4 : \begin{cases} a \to 00 \\ b \to 01 \\ c \to 011 \\ d \to 0111 \end{cases}$$

For $\mathcal{M}_3$, as soon as I receive or retrieve from file any of the codewords, we can proceed with decoding and immediately reconstruct the corresponding source symbol; conversely, with $\mathcal{M}_4$, if we receive/retrieve the codeword 01, we cannot proceed with immediate decoding, but w have to wait one or even two more bits until we see another 0. The first of the two codes is *instantaneous*, while the second is not. This feature of having little or no decoding delay is recommended in all so-called "real-time" applications, like remote driving.

Luckily enough, there is a necessary and sufficient condition to check whether a code is uniquely decodable and instantaneous, that is called the *prefix rule*: in the code, no codeword shall appear as a *prefix* (i.e., the first part) of any other word. Code $\mathcal{M}_3$ does not obey this condition: the word associated with b is in fact the prefix of two other words in the code table. A code that satisfies the prefix rule is called a *prefix* code. A fundamental result

regarding prefix (instantaneous) codes is *Kraft-McMillan inequality* regarding the length of the various codewords.

$$\mathcal{M}_2^{(2)}: \begin{cases} aa \rightarrow 00 \\ ab \rightarrow 010 \\ ac \rightarrow 011 \\ ad \rightarrow 000 \\ ba \rightarrow 100 \\ bb \rightarrow 1010 \\ bc \rightarrow 1011 \\ bd \rightarrow 1000 \\ ca \rightarrow 110 \\ cb \rightarrow 1110 \\ cc \rightarrow 1111 \\ cd \rightarrow 1100 \\ da \rightarrow 000 \\ db \rightarrow 0010 \\ dc \rightarrow 0011 \\ dd \rightarrow 0000 \end{cases}$$

Let us denote with L the number of symbols in the code alphabet, and let $l_m, m = 1, ..., M$ denote the lengths, measured in symbols, of the various codewords μ_m associated with the M source symbols. For M_3 we have $L = 2$, $l_1 = 2$, $l_2 = 2$, $l_3 = 3$, and $l_4 = 4$. Iff the codeword lengths $l_m, m = 1, ..., M$ obey the following inequality

$$\sum_{m=1}^{M} L^{-l_m} \leq 1 \tag{3.9}$$

then there exist a prefix (i.e., uniquely decodable instantaneous) code with codewords bearing such lengths.

Proving this is very in the case of binary codes, i.e., with $L = 2$. Suppose we have ordered the word lengths so that $l_1 \leq l_2 \leq ... \leq l_M$. We can display all possible codewords (of variable length) building a *binary tree* of order l_M as the one in Fig. 3.5. To find the node corresponding to any possible codeword, we start from the root and we follow a recursive rule, as shown in Fig. 3.5 (a) for $l_M = 5$: for each node, we add an "upward" branch and mark the child node with the parent node's word followed by a 1, and a "downward" branch marking the node with the parent node's word followed by a 0. The root word is null, so that in total the tree has 2^{l_M} terminal nodes (leaves).

This tree contains any possible codeword of any code, and it is particularly useful to visualize the prefix rule: if we want to construct a prefix code, we must ensure that no codeword has a child node of any order (i.e., at any distance toward the leaves of the tree) that is also a codeword. In the design of a code, if I choose a certain codeword of length l_m, the tree can no longer have all of the $2^{l_M}/2^{l_m} = 2^{l_M - l_m}$ leaves that derive from the chosen word, as shown in Fig. 3.5 (b) for word 01 with length 2 (in red): (all) $2^{5-2} = 8$ leaves (in gray) that have 01 as a prefix are deleted. Since I cannot delete more leaves from the tree than there are in total, repeating the reasoning for each codeword, it must happen that

$$2^{l_M - l_1} + 2^{l_M - l_2} + ... + 2^{l_M - l_M} = \sum_{m=1}^{M} 2^{l_M - l_m} \leq 2^{l_M} \tag{3.10}$$

Reducing, we immediately get Kraft's inequality – generalization to any L (Kraft-McMillan)is straightforward – showing the necessity of the condition. By inverse reasoning we can also proof sufficiency.

The meaning of Kraft's inequality is clear: if we wish to build a prefix code, the codewords cannot be of any arbitrary length. In particular, they cannot be too short, otherwise the count $\sum L^{-l_m}$ rapidly becomes greater than 1. In a moment, we will unfortunately see that on the contrary we would just like to build codes with as many short codewords as possible to make if as efficient as possible.

Once the notion of a code is well-posed, let us return to the fundamental issue of source compression. From the example of the run-length encoding algorithm, we have see that

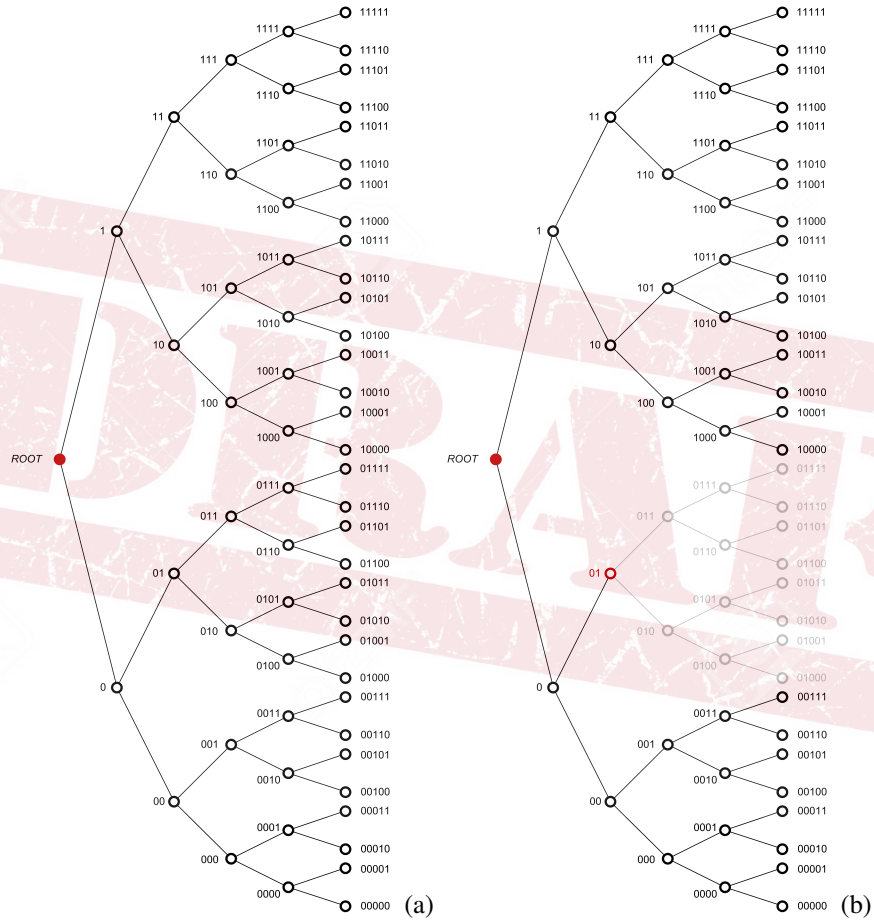


Figure 3.5 Kraft tree

some bits of the digital source are to be considered *redundant* since the string can be shortened. Can we try to somehow eliminate the *redundancy* of a source by avoiding transmitting that fraction of "useless" symbols and using the link or storage medium as efficiently as possible? The solution to the problem lies in an appropriate *lossless source coding* scheme. A naive criterion for choosing (designing) this code may be minimizing the length of its codewords, so that the encoded file has the smallest possible size and/or the digital link for real-time transmission requires the minimum bit rate.

If we consider a "long" string of N source symbols, N very large, to be stored in a file and we use a lossless binary code to encode that string, by the law of large number the size of the file in bits will be equal to $N \cdot l_{avg}$ where l_{avg} is the *avergae codeword lenght,*

$$l_{avg} \triangleq \sum_{m=1}^{M} p_m l_m \tag{3.11}$$

expressed in binary digits that are, on average, associated with each source symbol. A code with the minimum average word length requires the minimum transmission capacity from the network to communicate the information from a given source. If the source transmits at R_s symbols/s, the average transmission rate after encoding is $R_s^{cod} = R_s l_{avg}$ binary digits/s. We see that l_{avg} depends of course on the characteristics (word lengths l_m) of the code, but also on those *of the source* through the probabilities p_m of the different source symbols.

Two questions now arise:

1. Given a certain source, what is the minimum average length that we can have ?
2. How can we attain it ?

In particular, Kraft inequality 3.9 suggests that it is not possible to arbitrarily reduce the codewords lengths for a prefix code – there seems to be a "physical limit" below which it is not possible to go. To answer this question, we resort to the powerful framework of Shannon information theory.

3.3 Shannon Information

In the previous paragraph, we freely used terms such as *information*, *information rate*, *capacity* without providing a precise definition. We now overcome this ambiguity and clarify all these terms and concepts.

The notion of information arises from trying to establish the degree of *uncertainty* that is removed by the occurrence of a certain event. We can model a soccer match between Juventus and Pontedera as a random experiment having three possible outcomes: {Juventus Wins}, {Draw}, and {Pontedera Wins}. What is the degree of uncertainty, and how much information does the occurrence of any of the possible outcomes provide? The event {Juventus Wins}, which has the highest probability of occurrence, provides very little information because it removes very little uncertainty, and perhaps it isn't even given a sports feature in national newspapers or in socials. If on the other hand, Pontedera wins, {Pontedera Wins} provides *great* information and perhaps even makes the front page.

The information about a certain event A, which we will denote by $\mathrm{In}(A)$, is a *measure of the degree of removed uncertainty* by the occurrence of that event. For convenience and consistency with this empirical definition, $\mathrm{In}(A)$ must obey certain rules:

- $\text{In}(A) \geq 0$
- $\text{In}(A)$ changes inversely to $\Pr(A)$
- $\text{In}(A) = 0$ when $\Pr(A) = 1$, i.e., certainty brings no information
- If A_1 and A_2 are independent events, then $\text{In}(A_1 \cap A_2) = \text{In}(A_1) + \text{In}(A_2)$

A convenient definition that has these properties (especially the third one) is the following:

$$\text{In}(A) \stackrel{\triangle}{=} \log\left(1/\Pr(A)\right) = -\log\left(\Pr(A)\right) \tag{3.12}$$

The base of logarithms is not essential. If we use base 2, as is now universal practice, information is measured in *bit* (binary unit). In the early days of information theory, natural logarithms were used, and the unit was called *nat*. Therefore:

$$\text{In}(A) \stackrel{\triangle}{=} -\log_2\left(\Pr(A)\right) \tag{3.13}$$

By estimating the number of brides or grooms who say "NO" at the altar at 1 in 10,000, the reader can derive the number of bits of information contributed by this event (which sometimes deserves the front page of the local newspaper).

Example 3.12

Let us prove the necessity of the logarithmic measure of information according to the four properties above. For two independent events A and B with probability $\Pr(A) = p_A$ and $\Pr(B) = p_B$, we know that the yet-to-be-determined mathematical law μ that links probability to the information measure must feature the property $\mu(p_A \cdot p_B) = \mu(p_A) + \mu(p_B)$. We differentiate this relation member-by-member with respect to p_A:

$$p_B \cdot \mu'(p_A \cdot p_B) = \mu'(p_a) \tag{3.14}$$

where $\mu'(p) \stackrel{\triangle}{=} d\mu(p)/dp$. Differentiating now (3.14) *wrt* p_B:

$$\mu'(p_A \cdot p_B) + p_B p_A \cdot \mu''(p_A \cdot p_B) = 0 \;\Rightarrow\; \mu'(p) + p\mu''(p) = 0 \;\Rightarrow\; \big(p\mu'(p)\big)' = 0 \tag{3.15}$$

that is, $p\mu'(p) = \gamma$, where γ is any constant. We add the boundary condition $\mu(1) = 0$ (third property a) and solve the differential equation:

$$p\frac{d\mu}{dp} = \gamma \;\Rightarrow\; d\mu = \gamma\frac{dp}{p} \;\Rightarrow\; \int_1^p d\mu = \gamma\int_1^p \frac{dp}{p} \tag{3.16}$$

and, keeping in mind the boundary condition, we find the solution $\mu(p) = \gamma\ln(p)$. Taking into account that $0 \leq p \leq 1$ and the first property, $\gamma < 0$ can be arbitrary. The final solution is then $\mu(p) = -\log(p)$ where the log can be in any basis.

3.4 Information measure of a memoryless source

It is now simple to calculate the average amount of information provided by the generation of a single source symbol:

$$H(S) \triangleq \sum_{m=1}^{M} p_m \mathrm{In}(s_m) = \sum_{m=1}^{M} p_m \log\left(\frac{1}{p_m}\right) = -\sum_{m=1}^{M} p_m \log_2(p_m) \tag{3.17}$$

This *average information* value, which is also called *entropy* of the source, characterizes the average information content of a single symbol of S, and is uniquely determined by the probability law of the source itself.

If S generates symbols at a rate of R_s baud (symbols/s), the *information rate* (entropy rate) of the source is $R_b = R_s \cdot H(S)$ bits/s. Claude Shannon gave the name entropy to this measure of information, which he formalized in the 1940s after Hartley's pioneering studies in the 1920s. The term refers to the concept of entropy of a physical system. In statistical mechanics, Boltzmann discovered that thermodynamic entropy is proportional (through Boltzmann's constant k) to the logarithm of the number N of states that the physical system can occupy. If the states are equally probable, N is also equal to the inverse of the occupancy probability of each state $1/N$, consistent with Shannon's definition. Moreover, if the states are equally probable, the system is in the state of maximum disorder and, from basic physics, entropy is maximum: this quantity gives an indication of the degree of *disorder* or *uncertainty* of the system.

Consider now a particularly simple case: the memoryless *binary source* with $s_1 = 0$ and $s_2 = 1$. It is characterized by the probability p of the symbol 0, $\Pr(0) = p$, which also determines $\Pr(1) = 1 - p$. The entropy calculation is straightforward:

$$H(S) = p \log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{1}{1-p}\right) \triangleq \eta(p) \tag{3.18}$$

In the following, we will also use the property (easy to derive)

$$\frac{d\eta(p)}{dp} = \eta'(p) = \log\left(\frac{1-p}{p}\right) \quad 0 \le p \le 1 \tag{3.19}$$

From the plot of $\eta(p)$ reported in Fig. 3.6, we notice some properties of the entropy of a binary source, which can be easily extended to any M-ary source:

- $H(S) \ge 0$ (trivial);
- $H(S) = 0$ if and only if one of the probabilities of the symbols is equal to 1. In this case, the source always emits that symbol with certainty, and therefore does not provide any information content.;
- $H(s)$ It is invariant under permutations of the probabilities of the symbols – from this standpoint it enjoys a symmetry property.;
- $H(S) \le \log(M)$ and equality only holds if all the probabilities of the symbols are equal (and are therefore equal to 1/M);

The equiprobable source is the one with the greatest information content: any imbalance in the probabilities of symbols leads to a decrease in entropy. In fact, if the probabilities

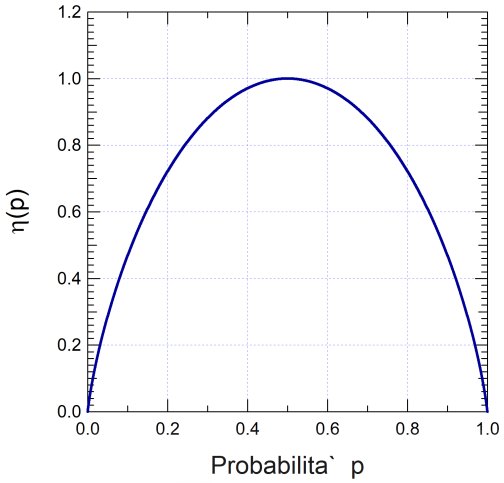


Figure 3.6 Entropy of the bimary source

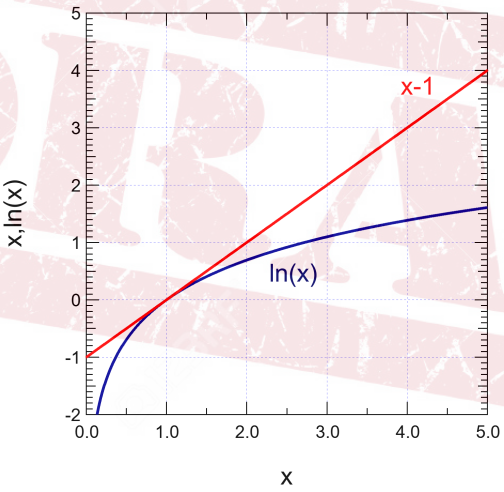


Figure 3.7 Fundamental inequality (leading to Gibbs')

are unbalanced, some symbols are more "certain' than others, and the reduced amount of information for each of these leads to a global decrease in average information. To prove the property that $H(S) \leq \log(M)$, we must derive a small "lemma" on discrete probability distributions known as *Gibbs' inequality*. If I have two probability laws p_m and q_m, $m = 1, \ldots, M$, then

$$\sum_{m=1}^{M} p_m \log\left(\frac{1}{p_m}\right) \leq \sum_{m=1}^{M} p_m \log\left(\frac{1}{q_m}\right) \tag{3.20}$$

In fact, it is well known from calculus that $\ln(x) \leq (x-1)$ (Fig. 3.7) and that the equality holds only for $x = 1$, where the two curves are tangent. Let $x = q_m/p_m$; it follows that $\ln(q_m/p_m) \leq (q_m/p_m - 1)$. If we multiply both sides of this relation by p_m and add over

m, we get

$$\sum_{m=1}^{M} p_m \ln\left(\frac{q_m}{p_m}\right) \leq \sum_{m=1}^{M} (q_m - p_m) \;\Rightarrow$$

$$\sum_{m=1}^{M} p_m \ln\left(\frac{q_m}{p_m}\right) \leq 0 \;\Rightarrow$$

$$\sum_{m=1}^{M} p_m \log\left(\frac{q_m}{p_m}\right) \leq 0 \tag{3.21}$$

where the right-hand side clearly vanishes because the sum of the two probability distributions is 1 for both, and the sign of the right-hand side does not change if we choose any base for the logarithms. The final step is then trivial.

Example 3.13

If we have

$$\sum_{m=1}^{M} p_m \log\left(\frac{q_m}{p_m}\right) \leq 0$$

then

$$\sum_{m=1}^{M} p_m \log\left(\frac{p_m}{q_m}\right) \geq 0$$

This second expression is used in statistics to evaluate how closely a certain probability law $\{p_m\}$ approaches a prototype law $\{q_m\}$ and is referred to as the *Kullback-Leibler divergence*:

$$D_{KL}(p||q) \triangleq \sum_{m=1}^{M} p_m \log\left(\frac{p_m}{q_m}\right) \geq 0 \tag{3.22}$$

The greater the divergence, the greater the deviation from the prototype law $\{q_m\}$ - divergence 0 means, conversely, identity.

For example, from experimental data we could have derived the two probability laws $p_m^{(1)}$ and $p_m^{(2)}$ for the sources with $M = 8$ levels shown in columns 1-2 in Table 3.1, and we would like to establish which of the two comes closest to a Bernoulli law with probability a 0.1-0.9

$$q_m = \binom{7}{m-1}(0.1)^{m-1}(0.9)^{7-m} \quad , \quad m = 1, \ldots, 8$$

whose values ere reported in column 3. Our data are also shown graphically in Fig. 3.8, where we see that the probability law closest to Bernoulli's is clearly law 1, as effectively indicated by the divergence in Table 3.1.

The K-L divergence is not a "distance", in the sense that it does not enjoy the property of symmetry ($D(p||q) \neq D(q||p)$), but it is effective as a global measure of deviation, as shown in this example.

How do we use this "lemma" to prove that $H(S) \leq \log(M)$? We let $q_m = 1/M$ for all m (i.e., the distribution of an equiprobable source). In Gibbs' inequality, the left-hand side

m	q_m	$p_m^{(1)}$	$p_m^{(2)}$
1	$1 \cdot 10^{-7}$	0,002413931	0,094056733
2	0,0000063	0,046366715	0,046978491
3	0,0001701	0,038960623	0,078982638
4	0,0025515	0,043802808	0,093429256
5	0,0229635	0,026206271	0,057075785
6	0,1240029	0,166344282	0,036994862
7	0,3720087	0,395708934	0,454060442
8	0,4782969	0,280196436	0,138421793
$D(p\|\|q)$	0	1,010426223	3,54912057

Table 3.1 Leggi di probabilità e Divergenza di K-L

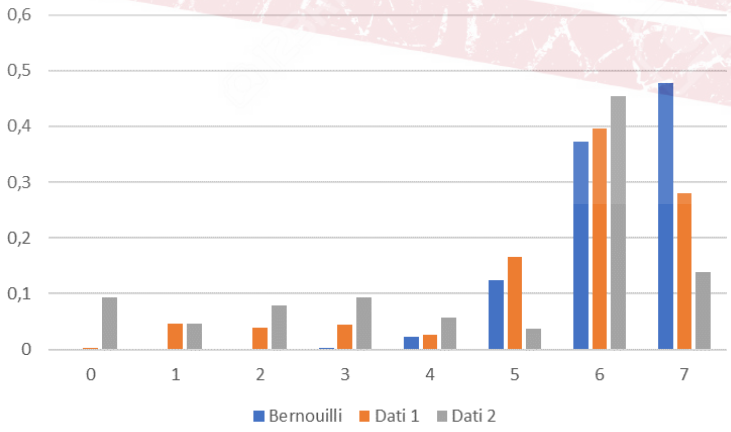


Figure 3.8 Probability laws

is in fact the entropy $H(S)$, so we have

$$H(S) \leq \sum_{m=1}^{M} p_m \log(M) = \log(M) \sum_{m=1}^{M} p_m = \log(M) \tag{3.23}$$

An M-ary source carries a maximum of $\log(M)$ bits/symbol, and produces information at a maximum rate of $R_b = \log(M) \cdot R_s$ bits/s. A generic memoryless source has in general an information content no greater than this, and can be characterized by a parameter ρ, $0 \leq \rho \leq 1$, called *redundancy*:

$$\rho \triangleq 1 - H(S)/\log(M) \tag{3.24}$$

The meaning of ρ is clear for the binary source. If the symbol rate is R_s binary symbols per second, its actual information rate is $R_b = H(S) \cdot R_s$ bits/s, while the maximum information rate it could support is $R_{b,M} = H_{\max} \cdot R_s = \log(2) \cdot R_s = R_s$ bits/s. Therefore

$$\frac{R_b}{R_{b,M}} = \frac{H(S) \cdot R_s}{R_s} = H(S) = 1 - \rho \tag{3.25}$$

Of all the binary source symbols, only the fraction $H(S) = 1 - \rho$ actually carries information, while the remaining fraction ρ is redundant, that is, useless for the purposes of transmitting information. A source with highly unbalanced probabilities has $H(S) \ll H_{\max}$, and is therefore highly redundant (ρ close to one). Conversely, a symmetric source has zero redundancy. From this observation, the concept of *compression* of a source arises naturally: if $H(S) < H_{\max}$, some of the source symbols appear "useless" for the transmission or storage of information, and it is perhaps possible to find a way to "remap" the source symbols so as to process the fraction ρ of redundant symbols.

To perform best the function of data compression, we can collect several consecutive symbols generated by the source into *blocks* of k symbols, and consider such blocks as produced by a *new* fictitious *extended* source to the order k. This new source is denoted by $S^{(k)}$, and its alphabet is made up of the M^k distinct, ordered blocks of k M-ary symbols that we can form. For example, in the case of the eighth extension of a binary source, the bits are grouped 8 by 8 into bytes, and the alphabet is, with obvious notation,

$$\Omega^{(8)} = \underbrace{\{00000000, 00000001, 00000010, ..., 11111110, 11111111\}}_{256 \text{ valori}} \tag{3.26}$$

Since the various symbols emitted by the original memoryless source are independent, the probability distribution of $S^{(k)}$ is easy to obtain (an exercise for the reader). In the example above, if $\Pr(0) = p$ and $\Pr(1) = q = 1 - p$, the probabilities are

$$\begin{array}{cccccccc} s_m^{(8)} & 00000000 & 00000001 & 00000010 & \dots & 11111110 & 11111111 \\ p_m^{(8)} & p^8 & p^7(1-p) & p^7(1-p) & \dots & p(1-p)^7 & (1-p)^8 \end{array} \tag{3.27}$$

Given the independence and considering the logarithmic definition of information, it is easy as well show that the entropy of the extended source is given by

$$H\left(S^{(k)}\right) = \underbrace{H(S) + H(S) + ... + H(S)}_{k \text{ volte}} = k \cdot H(S) \tag{3.28}$$

We will see in the next section how efficient compression needs extension of the source.

3.5 Optimum codes for a memoryless source

Consider the usual Martian language source $\Omega = \{a, b, c, d\}$ with the following symbol probabilities:

$$\begin{aligned} \Pr(a) &= 1/2 \\ \Pr(b) &= 1/4 \\ \Pr(c) &= 1/8 \\ \Pr(d) &= 1/8 \end{aligned} \tag{3.29}$$

The entropy for this source is $H(S) = -(1/2)(-1)-(1/4)(-2)-(1/8)(-3)-(1/8)(-3) = 1.75$ bit/simbolo. We can use the following binary prefix code (obeying Kraft's inequality)

$$\begin{aligned} a &\rightarrow 0 \\ b &\rightarrow 10 \\ c &\rightarrow 110 \\ d &\rightarrow 111 \end{aligned} \tag{3.30}$$

with an average lenght

$$l_{avg} = (1/2)(1) + (1/4)(2) + (1/8)(3) + (1/8)(3) = 1.75 \text{ bit} \tag{3.31}$$

For this specific example, we see that the average codeword length (in binary digits) is exactly equal to the entropy, i.e., the information content of the source (in bits) – it appears that we use just that necessary and sufficient number of bits that is needed to represent all of the information content of the source with no loss and no redundancy left. Is this a general rule? Can we get something even better than entropy? Not by chance, our code satisfies Kraft with equality...

If we believe that entropy represents the information content of the source, it is reasonable to assume that no binary prefix code can get to an average length (in binary digits) better than that. Can we confirm this conjecture? Let us define a *fictitious* porbability law as follows:

$$q_m \triangleq \frac{2^{-l_m}}{\sum_{m=1}^{M} 2^{-l_m}} = \frac{2^{-l_m}}{\Gamma} \tag{3.32}$$

Is is apparent that for any m $q_m > 0$, and that the sum of all probabilities is 1. Usig Gibb's inequality we get

$$H(S) \leq \sum_{m=1}^{M} p_m \log\left(\frac{1}{q_m}\right) = \sum_{m=1}^{M} p_m \left[\log\left(2^{l_m}\right) + \log(\Gamma)\right] \tag{3.33}$$

or

$$H(S) \leq \sum_{m=1}^{M} p_m l_m + \log(\Gamma) = l_{avg} + \log(\Gamma) \tag{3.34}$$

Since our code follows the prefix rule, Kraft's inequality says that $\Gamma \leq 1$, so that $\log(\Gamma) \leq 0$, therefore $l_{avg} \geq H(S)$ as stated above. If we use a generic L-ary, code alphabet, we can show that $l_{avg} \geq H(S)/\log(L)$.

The conclusion is that there indeed exist a "physical" boundary to the average length (i.e., the efficiency) of a lossless decodable code. Is there a way to design (optimal) codes

whose average length is as close as possible to the source entropy? Let us compare the definitions of $H(s)$ and l_{avg}:

$$H(S) = \sum_{m=1}^{M} p_m \log\left(\frac{1}{p_m}\right) \quad , \quad l_{avg} = \sum_{m=1}^{M} p_m l_m \tag{3.35}$$

If we can manage to get $l_m = \log(1/p_m)$, then the optimal code is found... Unfortunately, this cannot be in general done since l_m is bound to be *integer*. We can try to get as close as possible by defining a code having wordlengths as follows:

$$l_m = \left\lceil \log\left(\frac{1}{p_m}\right) \right\rceil \quad , \quad m = 1, \ldots, M \tag{3.36}$$

Approximating by excess, we will for sure obey (3.34), but we still make l_{avg} as close as possible to the entropy $H(S)$ by construction.

Can we actually build a prefix code with suche lengths? From(3.36) we have

$$\log\left(\frac{1}{p_m}\right) \leq l_m < \log\left(\frac{1}{p_m}\right) + 1 \tag{3.37}$$

or,

$$2^{\log(p_m)-1} \leq 2^{-l_m} < 2^{\log(p_m)}$$

Considering the first two terms only and summing on m,

$$2^{-l_m} \leq p_m \;\Rightarrow\; \sum_{m=1}^{M} 2^{-l_m} \leq \sum_{m=1}^{M} p_m \; = 1 \tag{3.38}$$

These lengths therefore satisfy Kraft's inequality, and allow us to construct a prefix code as desired.

A code with this property is the *Shannon-Fano* code. To construct it, we begin by finding the word lengths as in (3.36). Then, we considers the those with the minimum length (imagine that there are ℓ such words) and we give to these a binary word that is the binary representation of the number $0, 1, 2, ..., \ell - 1$ in any order until the group is over (can also be just a single word). Within the group, the code is certainly prefix because the words are all the same length and all different from each other. Once this (small) group is over, we move on to the next larger length, and we build the relevant words as the union of a prefix and a suffix: the prefix will be same for all words, and will be the binary representation of ℓ; the suffix will be obtained through the usual numbering process as above, starting from 0 up to the end of the group – and so on until the words are over. Kraft's inequality guarantees that this procedure always works and we never run out of a "number" for any word.

Example 3.14

Let's build the Shannon-Fano code for the 5-symbol source below:

$$\begin{aligned} &\Pr(a) = 0.05 \\ &\Pr(b) = 0.25 \\ &\Pr(c) = 0.25 \\ &\Pr(d) = 0.15 \\ &\Pr(e) = 0.3 \end{aligned} \tag{3.39}$$

The entropy of this source is approximately 2.15 bits/symbol. From (3.36) we have

$$l_1 = 2,\ l_2 = 2,\ l_3 = 2,\ l_4 = 3,\ l_5 = 5$$

and the codewords are

$$\begin{cases} e\ :\ l_1 = 2 \rightarrow 00 \\ b\ :\ l_2 = 2 \rightarrow 01 \\ c\ :\ l_3 = 2 \rightarrow 10 \\ d\ :\ l_4 = 3 \rightarrow 110 \\ a\ :\ l_5 = 5 \rightarrow 11100 \end{cases}$$

The average length is also

$$l_{avg} = 0.3 \cdot 2 + 0.25 \cdot 2 + 0.25 \cdot 2 + 0.15 \cdot 3 + 0.05 \cdot 5 = 2.3 \text{ bit/simbolo}$$

How efficient is this code? Can we evaluate its general performance in terms of l_{avg}? Starting again from (3.36), this time multiplying all the members by p_m and summing over m (that is, calculating expected values), we get:

$$\sum_{m=1}^{M} p_m \log\left(\frac{1}{p_m}\right) \leq \sum_{m=1}^{M} p_m l_m < \sum_{m=1}^{M} p_m \left(\frac{1}{p_m}\right) + \sum_{m=1}^{M} p_m \tag{3.40}$$

that is

$$H(S) \leq l_{avg} < H(S) + 1 \tag{3.41}$$

so that, the average length of the Shannon-Fano code approximates entropy to within one bit: we have derived a bound on the code performance. For the Shannon-Fano code example we just saw, the source entropy is 2.15 bits and $l_{avg} = 2.3$ binary digits.

Is there a better code than Shannon-Fano's? To answer this, we construct a tree with M leaves as follows. We begin by sorting the source symbols in decreasing probability and assign those probabilities to the M leaves of the tree . We then proceed to merge the two smaller probabilities by converging the two branches of the tree that originate from such leaves into a single node, which we label with the sum probability. We thus obtain a set of $M - 1$ nodes to which we apply (recursively) the same procedure to obtain a set of $M - 2$ nodes, and so on. Looking at the tree, we see that starting from the original $M - ary$ source S, at the i-th reduction step, with i flowing in a reverse fashion $i = M - 1, \ldots, 1$, a new "reduced source" with i symbols is created that we can call S_i, and that, compared to the one in the previous step S_{i+1}, has a number of symbols reduced by 1 (where $S_M \stackrel{\triangle}{=} S$). In Fig. 3.9, the merging of a and b gives the node with probability 0.2, and so on. In reducing the source, we also mark the branches created in the process with the symbols 0 and 1, following a convention we will maintain throughout the construction: the branch coming out of the node with the lowest probability is labeled 1, the one coming out of the node with the highest probability is labeled 0. We iterate this procedure until we reach the root with unit probability ("source" S_1), obtaining the tree in Fig. 3.9.

Once this is done, we proceed to label the tree nodes, much as we have already seen in Fig. **??**: starting from the root, we "visit" the tree, that is, we proceed (backwards) from the root to the leaves (i.e., the source symbols). If we take an upward branch, we label the destination node with the word from the parent node, appending a 0 at the end; if we take

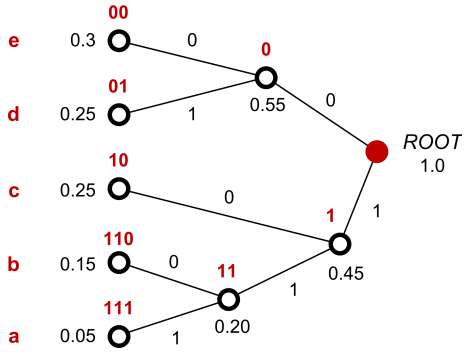


Figure 3.9 Example of Huffman coding

the downward branch, we add a 1. When we reach a leaf, we obtain the codeword that we associate to the corresponding source symbol.

This procedure is called *Huffman coding*, and it can easily be shown that ir generates a prefix code. The tree also contains the Huffman codes (generated with the same criteria) for all the "reduced", intermediate sources S_i. The iterative encoding algorithm is very simple, and is therefore efficient for real-time applications. The same is true for the decoding process, which can be reduced to an efficient tree search. The Huffman code is still used as an efficient lossless compression algorithm for example as a final so-called *entropy* code in the JPEG format to compress images.

The basic idea behind the construction of the Huffman (prefix) code is the usual one: short codewords are associated with the most probable (most frequent) symbols, while longer codewords are reserved for the less frequent symbols. This strategy clearly minimizes the average word length. In our example, we have

$$l_{avg} = 0.3 \cdot 2 + 0.25 \cdot 2 + 0.25 \cdot 2 + 0.15 \cdot 3 + 0.05 \cdot 3 = 2.2 \qquad (3.42)$$

very close to the theoretical lower limit (and better than the Shannon-Fano one). The reader may discuss the special case in which the source is such that an encoding is found having exactly $l_m = -\log(p_m)$, and give an example of such a source with its corresponding encoding.

The key property of the Huffman code is that it represents, given the source, the lossless prefix code that has, amidst all the possible ones, the *minimum* l_{avg} – we omit the proof. In practice, codes other than those just introduced are also used, for specific reasons. For instance, in the lossless audio encoding `.flac`, after a differential stage with prediction akin to that in section 3.1.3, the residual error $e[n]$ is *not* quantized on very few bits (as in voice DPCM); rather, it is encoded with the lossless Golomb-Rice code. Such algorithm has a very simple algorithmic implementation and does not require tables, so that it is advantageous for sources with a large M. On the other hand, it is optimal only for a particular behavior of the probability law of the source symbols (geometric law), which is encountered only in certain applications like audio.

3.5.1 Coding of the extended source

From our example, we see that Huffman code, although optimal, does *not* generally attain entropy. In addition, from a practical point of view, both Huffman and Shannon-Fano are

codes that are well suited to a multilevel source, but are incapable of encoding a binary source (the reader explain why). We discuss below how to solve these two issues by means of *extended source coding*.

Imagine that, given an information source, we have constructed its k-th order extension. We may design a Shannon-Fano code for this source. By choosing the length of the M^k words according to (3.36) and applying (3.41) to this new source, we have

$$H(S^{(k)}) \leq {l_{avg}}^{(k)} < H(S^{(k)}) + 1 \quad \Rightarrow \quad k \cdot H(S) \leq {l_{avg}}^{(k)} < k \cdot H(S) + 1 \quad (3.43)$$

where ${l_{avg}}^{(k)}$ is the average length of the words of the S-F code applied to the extended source, expressed in binary symbols for k original source symbols. Dividing by k we finally have

$$H(S) \leq {l_{avg}}^{(k)}/k < H(S) + 1/k \quad (3.44)$$

Since each codeword is associated with a block of k source symbols, the quantity ${l_{avg}}^{(k)}/k$ represents an "equivalent average length" for a single symbol of the original, unextended source, and is akin to the l_{avg} of th S-F single-symbol code (although numerically different in general). If k is sufficiently large, it can be seen from (3.44) that ${l_{avg}}^{(k)}/k$ approaches the source entropy with arbitrary accuracy. It is clear that if this property holds for the Shannon-Fano code, it must hold for the Huffman code as well, that guarantees a lower l_{avg} than that of the Shannon-Fano code for any extension order k.

The recipe for efficient encoding is simple: encode sources as extended as possible, and you will get arbitrarily close to the entropy. This statement, in a rather naive form, represents the so-called Shannon's first theorem of coding, or the data compression theorem. This approach is not just a theoretical trick – it is also used in the practice: in the LZW77 compressor that we will see later on, the binary string (file) to be compressed is considered *byte by byte*, i.e., adopting a source extension with $K = 8$.

Example 3.15

We consider a binary source with $p_1 = p = 0.1$ and $p_2 = q = 1 - p = 0.9$. The entropy of this source is $H(S) \simeq 0.469$ bits/symbol. In Fig. 3.10 we can see the evolution of the quantity $l_{avg}^{(k)}/k$ (as well as other quantities regarding source and code) for the Shannon-Fano code as k varies. As can be seen, $l_{avg}^{(k)}$ gets closer and closer to $H(S^{(k)})$ as predicted by (3.43).

The data compression theorem unfortunately clashes with the need to keep the coding/decoding delay low and, above all, to keep the complexity of the encoding algorithm sufficiently low. Extending a source to order k means exponentially increasing the number of distinct codewords and therefore the complexity of the codec (coder/decoder).

3.5.2 Arithmetic Coding

Another example of (asymptotically) optimum data compression is arithmetic coding. There are many variations of this technique, some of which are even patented, but there is no reference and/or standard implementation. Its advantage over the Huffman code is that it can (more) easily be made adaptive: the algorithm can start encoding the source string without any information about the probability law of the source itself.

S

	pm	-log(pm)	-pm log(pm)		lm	pm lm	
0	0,1	3,321928	0,3321928		4	0,4	
1	0,9	0,152003	0,1368028		1	0,9	
			0,4689956	H(S)		1,3	lmed
						1,3	lmed/k

$S^{(2)}$

	pm	-log(pm)	-pm log(pm)		lm	pm lm	
00	0,01	6,643856	0,0664386		7	0,07	
01	0,09	3,473931	0,3126538		4	0,36	
10	0,09	3,473931	0,3126538		4	0,36	
11	0,81	0,304006	0,246245		1	0,81	
			0,9379912	$H(S^{(2)})$		1,6	lmed(k)
			0,4689956	H(S)		0,8	lmed(k)/k

$S^{(3)}$

	pm	-log(pm)	-pm log(pm)		lm S-F	pm lm	
000	0,001	9,965784	0,0099658		10	0,01	
001	0,009	6,795859	0,0611627		7	0,063	
010	0,009	6,795859	0,0611627		7	0,063	
011	0,081	3,625934	0,2937007		4	0,324	
100	0,009	6,795859	0,0611627		7	0,063	
101	0,081	3,625934	0,2937007		4	0,324	
110	0,081	3,625934	0,2937007		4	0,324	
111	0,729	0,456009	0,3324308		1	0,729	
	1		1,4069868	$H(S^{(3)})$		1,9	lmed(k)
			0,4689956	H(S)		0,633333	lmed(k)/k

$S^{(4)}$

	pm	-log(pm)	-pm log(pm)		lm	pm lm	
0000	0,0001	13,28771	0,0013288		14	0,0014	
0001	0,0009	10,11779	0,009106		11	0,0099	
0010	0,0009	10,11779	0,009106		11	0,0099	
0011	0,0081	6,947862	0,0562777		7	0,0567	
0100	0,0009	10,11779	0,009106		11	0,0099	
0101	0,0081	6,947862	0,0562777		7	0,0567	
0110	0,0081	6,947862	0,0562777		7	0,0567	
0111	0,0729	3,777937	0,2754116		4	0,2916	
1000	0,0009	10,11779	0,009106		11	0,0099	
1001	0,0081	6,947862	0,0562777		7	0,0567	
1010	0,0081	6,947862	0,0562777		7	0,0567	
1011	0,0729	3,777937	0,2754116		4	0,2916	
1100	0,0081	6,947862	0,0562777		7	0,0567	
1101	0,0729	3,777937	0,2754116		4	0,2916	
1110	0,0729	3,777937	0,2754116		4	0,2916	
1111	0,6561	0,608012	0,3989169		1	0,6561	
	1		1,8759824	$H(S^{(4)})$		2,2037	lmed(k)
			0,4689956	H(S)		0,550925	lmed(k)/k

$S^{(5)}$

	pm	-log(pm)	-pm log(pm)		lm	pm lm	
00000	0,00001	16,60964	0,0001661		17	0,00017	
00001	0,00009	13,43972	0,0012096		14	0,00126	
00010	0,00009	13,43972	0,0012096		14	0,00126	
00011	0,00081	10,26979	0,0083185		11	0,00891	
00100	0,00009	13,43972	0,0012096		14	0,00126	
00101	0,00081	10,26979	0,0083185		11	0,00891	
00110	0,00081	10,26979	0,0083185		11	0,00891	
00111	0,00729	7,099865	0,051758		8	0,05832	
01000	0,00009	13,43972	0,0012096		14	0,00126	
01001	0,00081	10,26979	0,0083185		11	0,00891	
01010	0,00081	10,26979	0,0083185		11	0,00891	
01011	0,00729	7,099865	0,051758		8	0,05832	
01100	0,00081	10,26979	0,0083185		11	0,00891	
01101	0,00729	7,099865	0,051758		8	0,05832	
01110	0,00729	7,099865	0,051758		8	0,05832	
01111	0,06561	3,92994	0,2578434		4	0,26244	
10000	0,00009	13,43972	0,0012096		14	0,00126	
10001	0,00081	10,26979	0,0083185		11	0,00891	
10010	0,00081	10,26979	0,0083185		11	0,00891	
10011	0,00729	7,099865	0,051758		8	0,05832	
10100	0,00081	10,26979	0,0083185		11	0,00891	
10101	0,00729	7,099865	0,051758		8	0,05832	
10110	0,00729	7,099865	0,051758		8	0,05832	
10111	0,06561	3,92994	0,2578434		4	0,26244	
11000	0,00081	10,26979	0,0083185		11	0,00891	
11001	0,00729	7,099865	0,051758		8	0,05832	
11010	0,00729	7,099865	0,051758		8	0,05832	
11011	0,06561	3,92994	0,2578434		4	0,26244	
11100	0,00729	7,099865	0,051758		8	0,05832	
11101	0,06561	3,92994	0,2578434		4	0,26244	
11110	0,06561	3,92994	0,2578434		4	0,26244	
11111	0,59049	0,760015	0,4487815		1	0,59049	
	1		2,344978	$H(S^{(5)})$		2,58146	lmed(k)
			0,4689956	H(S)		0,516292	lmed(k)/k

Figure 3.10 Shannon-Fano code for a binary source extended up to order $k = 5$

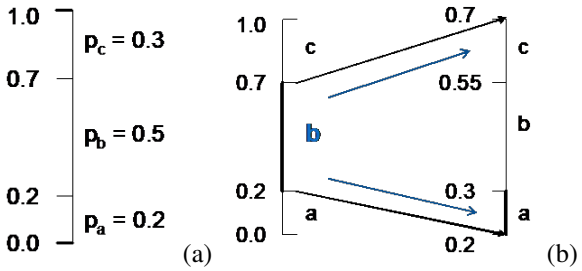


Figure 3.11 Arithmetic coding – 1st step

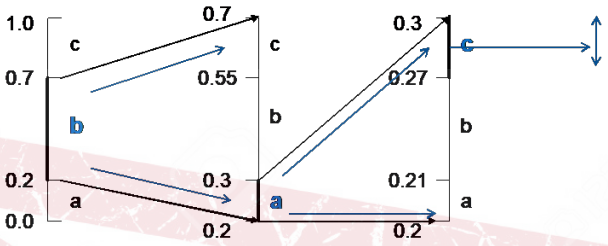


Figure 3.12 Arithmetic coding – 2nd step

Let us start from the non-adaptive version, for which the a priori source probabilities p_m are assumed to be known . The basic principle of encoding is simple: we try to associate with the source string a suitably constructed base-10 number β, the binary representation of which will then constitute the binary encoding of the original source string. To see how it is done, we make a simple example. Consider a memoryless ternary source with alphabet $\Omega \equiv \{s_0 = a, s_1 = b, s_2 = c\}$ and with probability distribution $p_0 = 0.2$, $p_1 = 0.5$, $p_2 = 0.3$. We then plot (vertically) the segment of the real axis (0,1) and split it into three adjacent intervals whose amplitude represents the probabilities of the given source, as shown in Fig. 3.11 (a). The intermediate points of this subdivision represent the values taken by the probability distribution (cumulative) function of the given source, which, since the source is discrete, has a "staircase" trend, piecewise constant and non-decreasing. Imagine now that the string to be encoded is bac. Since the first symbol is b, we consider the central interval of the segment (0,1) with width $\Delta_1 = p_b = 0.5$ (corresponding to the probability of b) and we "zoom" this interval subdividing it into M=3 parts with widths still proportional to the source probabilities, as shown in Fig. 3.11 (b). We then repeat the procedure: in the second step, we zoom in and consider the lower interval with amplitude $\Delta_2 = p_b p_a = 0.1$ since the second source symbol is a, and we conclude the encoding, after a further zoom and subdivision operation, considering the upper interval with amplitude $\Delta_3 = p_b p_a p_c = 0.03$ since the final symbol is c, obtaining the configuration in Fig. 3.12. As can be seen, we have identified a "final sub-interval" of the starting interval (0,1), in this case (0.27,0.3). The real number to associate with the source string is now the midpoint of the final interval: $\beta = 0.285$.

We now need to understand how to find the binary encoding of β. The problem is very simple: we take the first L significant digits (after the decimal point) of the binary representation of β, where $L = \lceil \log_2(1/\Delta_3) \rceil + 1$. In our case $L = 7$, $(\beta)_2 = 0.010010001...$ and the encoding is 0100100. The decoder applies a reverse processing. From the encoded string,

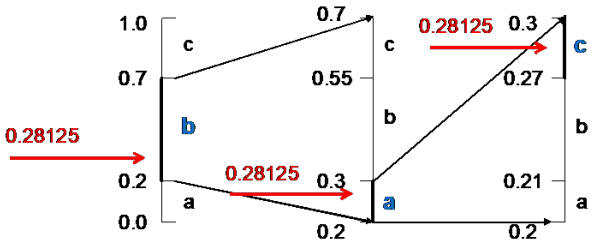


Figure 3.13 Arithmetic decoding

it can reconstruct a decimal real number $\bar{\beta}$, which represents the "truncation" of β as above. In our case, converting the string 0100100 gives $\bar{\beta} = (0.0100100)_2 = (1/4 + 1/32)_{10} = 0.28125$. It is easy to show (as an exercise) that even if $\bar{\beta} \neq \beta$, having truncated to L binary digits, $\bar{\beta}$ is still included within the final interval of the encoding procedure – the key point. The value $\bar{\beta}$ represents the "pointer" to the source string, and therefore decoding is immediate: the value 0.28125 in the first step points to the interval corresponding to b (see Fig. 3.13), in the second step, after the first zoom that the decoder must perform, it points to the first interval corresponding to a, and obviously in the third step, after the second zoom, it points to the upper interval, corresponding to c - end of decoding.

As with all source encoding algorithms, we must now discuss the compression factor and processing speed. The procedure that we have described, when implemented to work in real time, is rather slow because it is not based on a table, but on the real-time execution of an algorithm. That said, let's try to evaluate the compression factor. To do this, we will apply a resoning based oin the law of large numbers. If we think of encoding a very long source string, composed of N consecutive symbols with $N \to \infty$, then it is clear that due to the invariance of the source, the average code length in terms of binary digits/symbol can be evaluated as $l_{avg} = \lim_{N\to\infty} (L/N)$, that is, the total number L of binary digits encoding the entire string, divided by the number N of source symbols in the string.

In our code, we know that the final width Δ_N of the sub-interval encoding the string is $\Delta_N = \prod_{n=1}^{N} p_{m[n]}$ where $m[n] \in \{0, ..., M-1\}$ is the index of the n-th symbol of the source string. Consider now the quantity

$$\log(\Delta_N) = \sum_{n=1}^{N} \log(p_{m[n]}) \tag{3.45}$$

We can rearrange the summation by grouping all the terms that contain p_0 and which will be equal to a certain number N_0, all those, in number N_1, containing $p_1, \ldots$, all those, in number N_{M-1}, containing p_{M-1}, obtaining

$$\log(\Delta_N) = \sum_{n=1}^{N} \log(p_{m[n]}) = N_0 \log(p_0) + N_1 \log(p_1) + ... + N_{M-1} \log(p_{M-1}) \tag{3.46}$$

If N is very large, by approximating the probabilities with the relative frequencies, we easily obtain

$$\log\left(\frac{1}{\Delta_N}\right) = N \sum_{m=0}^{M-1} \frac{N_m}{N} \log\left(\frac{1}{p_m}\right) \cong N \sum_{m=0}^{M-1} p_m \log\left(\frac{1}{p_m}\right) = N \cdot H(S) \tag{3.47}$$

Now, the average code length per source symbol can be evaluated as

$$l_{avg} = \lim_{N\to\infty}\left(\frac{L}{N}\right) = \lim_{N\to\infty}\left(\frac{\lceil \log_2(1/\Delta_N)\rceil + 1}{N}\right) = \lim_{N\to\infty}\left(\frac{\lceil NH(S)\rceil}{N}\right) = H(S) \tag{3.48}$$

As is seen, the code asymptotically attains the entropy of the source! For finite length, an inefficiency term remains, as in the Shannon-Fano code, which, however, tends to vanish for long source strings.

As we have already mentioned, and unlike the Huffman and Shannon-Fano codes, it is not necessary to construct a code table to perform encoding: the code is algorithmic in the sense that the codeword is computed at runtime without consulting any table. The issue of latency would remain, because in our description, the encoded binary digit string is produced when the entire source string has been processed. In reality, practical implementations of arithmetic encoding produce the encoded binary digits at each zoom, that is, after receiving each new symbol, thus achieving real-time encoding with relatively low complexity.

3.5.3 Adaptive Arithmetic Coding

How can we make the arithmetic code adaptive, that is, effective even in the absence of a priori information about the source distribution? The answer is simple: starting with the hypothesis of a maximally uncertain source (i.e., with equally likely symbols) and gradually updating the presentation probabilities of the source symbols with which to perform the next encoding step.

Again, in the case of a ternary source, let's take the same example as the previous case of the string bac to be encoded. Fig. 3.14 (a) shows the state of the encoder at the first step and (b) shows what happens at the second step. As can be seen, from the first to the second step, the presumed probabilities of the symbols have changed to take into account the new presentation frequencies: at the initial step, the encoder starts by assuming that the symbols have appeared a number of times $N_a = N_b = N_c = 1$ for a total of $N_{tot} = 3$ extractions, so that they are equally likely. In the subsequent steps, the values of N_a, N_b, N_c, and N_{tot} are updated according to the actual step-by-step presentations of the various symbols: after the first step $N_a = N_c = 1$, $N_b = 2$, and of course $N_{tot} = 4$: this justifies the new values 1/4, 2/4, and 1/4 of the probabilities. The final state of the encoder is that of Fig. 3.15, from which it follows that $\beta = 49/120$.

Ignoring the (immaterial) truncation of $\bar{\beta}$, the operation of the decoder is clear: it must start with the equally probable subdivision shown in Fig. 3.14 (a), on which $\beta = 49/120$ clearly identifies b; the decoder, having observed b, updates the symbol probabilities accordingly and performs the first step as in Fig. 3.15 (b). In this new state, the value 49/120 leads to the decoding of a and to the consequent updating of the probabilities as in Fig. 3.15 (b); the last decoding step with 49/120 is immediate.

The adaptive encoder goes through a certain initial "training" phase in which the compression ratio is not optimal – this happens until the estimated source probabilities have stabilized around the actual values, after which the compression ratio is optimal.

3.6 Information sources with memory

In the speech coding example, we already introduced an example of a source with memory. The signal sampled at 4 kHz and represented as 8 bits can be considered as produced by

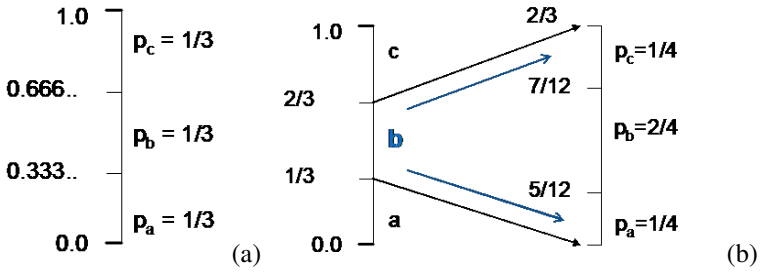


Figure 3.14 Adaptive Arithmetic Coding

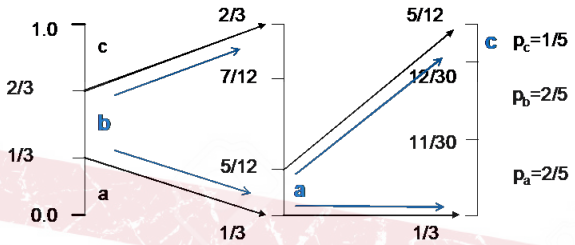


Figure 3.15 Adaptive Arithmetic Decoding

a source with M = 256 symbols (the possible values of the 8 bits) in which the various symbols are not independent. A sampled signal with no memory cannot represent voice but it is actually... white noise!

Another example is given by image coding. Consider the image in Fig. 3.16 (a) of the *Lena* model universally adopted for image processing examples. It consists of 256x256 pixels, each with 256 gray levels. Assuming we scan the image starting from the top-left pixel and proceeding from left to right and from top to bottom, we obtain a string of 65,536 symbols belonging to an alphabet with $M = 256$ values representing the output of our information source $S[n]$. Especially in the background areas, the values of two adjacent pixels are generally very similar, therefore there will be in general a high statistical correlation between the source symbols. In this case, too, the source has a certain amount of memory *memory-enabled* and this feature can be leveraged to attempt to encode it optimally. For comparison, we see in Fig. 3.16 (b) what an equivalent memory-free source looks like: noise again! Note that, in the case of image encoding, $H_{\max}$=8 bits/symbol. Efficient image encoding algorithms attain an average length l_{avg} even less than one bit per symbol, achieving very high source compression ratios.

The general case of a source with memory is complicated, because a complete characterization of such a source would theoretically require knowledge of the family of joint probabilities.

$$p_{m_1,m_2,...,m_N} \triangleq \Pr\{S[n] = s_{m_1}, S[n-1] = s_{m_2}, ..., S[n-N+1] = s_{m_N}\} \quad (3.49)$$

for any value of N. Alternatively, it would be necessary to know the first-order description, as for the memoryless source, plus the family of conditional probabilities

$$\Pr\{S[n] = s_{m_1} | S[n-1] = s_{m_2}, ..., S[n-N+1] = s_{m_N}\} \quad (3.50)$$

for any N.

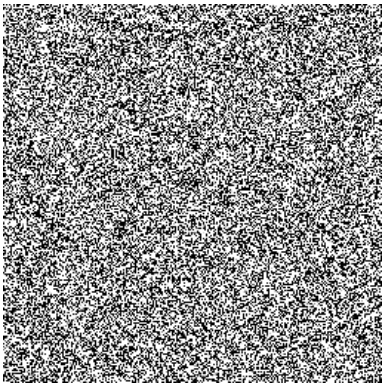

(a)

(b)

Figure 3.16 Examples of information source: (a) with memory, (b) memoryless

Since we speak of conditional probabilities, let us focus on the notion of *conditional information*: what is the information conveyed at time n by the symbol s_i upon observing the symbol s_k at time $n-1$? That is, what is the information of the event $\{S[n] = s_i | S[n-1] = s_k\}$? The answer is simple: abbreviating the notation,

$$\mathrm{In}\,(s_i|s_k) = \log\left(\frac{1}{\Pr(s_i|s_k)}\right) \tag{3.51}$$

This also leads to the definition of *joint information* of two symbols:

$$\mathrm{In}\,(s_i, s_k) = \log\left(\frac{1}{\Pr(s_i, s_k)}\right) = \log\left(\frac{1}{\Pr(s_i|s_k)\Pr(s_k)}\right) \tag{3.52}$$

so that

$$\mathrm{In}\,(s_i|s_k) = \mathrm{In}\,(s_i, s_k) - \mathrm{In}\,(s_k) \tag{3.53}$$

The conditional information is equal to the joint information of the two symbols minus the information carried by the previous symbol s_k which, having been observed and therefore known with certainty, no longer (a posteriori) carries any information. If the source is memoryless, the symbols are independent, and the conditional probability $\Pr(s_i|s_k)$ is equal to the marginal probability $\Pr(s_i)$. In this case, the conditional information is equal to $\mathrm{In}\,(s_i)$ and we find the familiar relation that, for independent events, the joint information is equal to the sum of the marginal informations.

If we consider the average information in (3.53), that is, averaging over the joint second-order probabilities $p(s_i, s_k)$ of the source, we get

$$\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i, s_k)\log\left(\frac{\Pr(s_k)}{\Pr(s_i, s_k)}\right) =$$

$$\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i, s_k)\log\left(\frac{1}{\Pr(s_i, s_k)}\right) - \sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i, s_k)\log\left(\frac{1}{\Pr(s_k)}\right) \tag{3.54}$$

On the right-hand side, we can easily recognize $H\,(S[n], S[n-1])$ and $-H\,(S[n-1])$: the summation over i of the second term marginalizes the joint probability and gives the

result $\Pr(s_k)$. Let's work on the right-hand side instead:

$$\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i,s_k)\log\left(\frac{\Pr(s_k)}{\Pr(s_i,s_k)}\right)$$

$$=\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i,s_k)\log\left(\frac{\Pr(s_k)\Pr(s_i)}{\Pr(s_i,s_k)}\right)+\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i,s_k)\log\left(\frac{1}{\Pr(s_i)}\right) \quad (3.55)$$

As above, the second term on the right-hand side is simply $H\left(S[n]\right)$, and so we can write

$$H\left(S[n],S[n-1]\right)$$

$$=H\left(S[n]\right)+H\left(S[n-1]\right)-\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i,s_k)\log\left(\frac{\Pr(s_i,s_k)}{\Pr(s_k)\Pr(s_i)}\right) \quad (3.56)$$

The additional term is called *mutual information* between $S[n]$ and $S[n-1]$ and clearly measures the degree of (in)dependence between the two values of the source. It is easy to see that if the source is memoryless, we have trivially $H\left(S[n],S[n-1]\right)=H\left(S[n]\right)+H\left(S[n-1]\right)$, otherwise the joint entropy is always less than the sum of the marginals because the mutual information

$$I\left(S[n],S[n-1]\right)\triangleq\sum_{i=1}^{M}\sum_{k=1}^{M}\Pr(s_i,s_k)\log\left(\frac{\Pr(s_i,s_k)}{\Pr(s_k)\Pr(s_i)}\right) \quad (3.57)$$

is always nonnegative, as a consequence of Gibbs' inequality (3.20). This implies that a string from a source with memory generally carries less information than a string of the same length from a memoryless source with the same marginal statistics; the memory source is therefore more compressible than a memoryless one.

The general calculation of the entropy of a source with a memory of arbitrary length, and therefore the characterization of its information content, is simple in principle but extremely complex in practice. Considering invariant sources, what we need to do is consider the k-th extension $S^{(k)}$ of the source and calculate its entropy:

$$H(S^{(k)})=H(S[n],S[n-1],...,S[n-k+1])=$$

$$\sum_{m_1,m_2,...,m_k=1}^{M}p_{m_1,m_2,...,m_k}\log\left(\frac{1}{p_{m_1,m_2,...,m_k}}\right) \quad (3.58)$$

Dividing by the extension length k we obtain a per-symbol entropy equivalent (let's say commensurate) to that of a single-symbol source: the correct measure of information content for a source with memory is nothing but the limit $k\to\infty$ of this per-symbol entropy:

$$H(S)\triangleq\lim_{k\to\infty}\frac{H(S^{(k)})}{k} \quad (3.59)$$

and it is called *entropy rate*. If the source is memoryless, $H(S^{(k)})=kH(S)$ and we fall back to the already known definition.

3.6.1 Markov Sources

In many cases, we can approximate the memory characteristics of a certain source modeling it as a Markov chain, i.e., a Markov process of order 1, in which knowledge of the entire "history" of the source is equivalent to knowledge of the only symbol immediately preceding the current one. In this case, conditioning on source values that are more than one step away is unnecessary, and the chain is completely characterized by the *transition probabilities matrix*.[1]

$$\mathbf{\Pi} = \{p_{ik}\} \quad , \quad p_{ik} \stackrel{\Delta}{=} \Pr\{S[n] = s_i | S[n-1] = s_k\} \tag{3.60}$$

In fact, joint statistics of any order can be obtained from the marginal of order 1 and from $\Pr\{S[n] = s_i | S[n-1] = s_k\}$ (briefly, $p(s_i|s_k)$):

$$p(s_{i_1}, s_{i_2}, \ldots, s_{i_N}) = p(s_{i_1}|s_{i_2}) \Pr(s_{i_2}|s_{i_3}) \ldots p(s_{i_{N-1}}|s_{i_N}) p(s_N) \tag{3.61}$$

We see that we have made the hypothesis of transition probabilities that do not depend on time n, that is, of a *homogeneous* or *stationary* chain.

From the transition matrix, we can calculate the so-called asymptotic probabilities of the individual states, that is, the probabilities of the presentation of the various symbols after the chain has operated for a long time. By collecting these probabilities in the vector $\mathbf{p} \stackrel{\Delta}{=} [p_1, p_2, ..., p_M]^T$, we must have

$$\mathbf{\Pi p} = \mathbf{p} \text{ con il vincolo } \sum_{m=1}^{M} p_i = 1 \tag{3.62}$$

This ensures that the probability distribution of the states remains constant over time, after an initial transient phase, depending on the initial state.

Understanding whether and how it is possible to encode a source with memory more efficiently than one without memory is easy. Consider a ternary Markov source with alphabet $\{a, b, c\}$ and transition matrix

$$\mathbf{\Pi} = \begin{bmatrix} 1/3 & 1/4 & 1/4 \\ 1/3 & 1/2 & 1/4 \\ 1/3 & 1/4 & 1/2 \end{bmatrix} \tag{3.63}$$

The transition probabilities are calculated by solving

$$\begin{cases} \frac{1}{3}p_a + \frac{1}{4}p_b + \frac{1}{4}p_c = p_a \\ \frac{1}{3}p_a + \frac{1}{2}p_b + \frac{1}{4}p_c = p_b \\ \frac{1}{3}p_a + \frac{1}{4}p_b + \frac{1}{2}p_c = p_c \\ p_a + p_b + p_c = 1 \end{cases} \Rightarrow \begin{cases} -\frac{2}{3}p_a + \frac{1}{4}p_b + \frac{1}{4}p_c = 0 \\ \frac{1}{3}p_a - \frac{1}{2}p_b + \frac{1}{4}p_c = 0 \\ p_a + p_b + p_c = 1 \end{cases} \tag{3.64}$$

which, as indicated, can indeed be solved because the third equation is a combination of the first two and can be eliminated. The solution is

$$p_a = \tfrac{3}{11} \ , \ p_b = p_c = \tfrac{4}{11} \tag{3.65}$$

[1]The chain must be ergodic, meaning that there cannot be any process state that, if reached, prevent the subsequent reaching of some other states. This guarantees that the chain attains all allowed states with a certain frequency and stabilizes on an asymptotic probability distribution (see text).

Based on this result, we can use the following encoding strategy: if the source is in state a, to encode the next symbol we use a Huffman code based on the transition probabilities from state a, which represent the probabilities of a symbol conditioned on the source being in state a. The same applies for states b and c. Summarizing this strategy state by state, we have:

Stato a:

$$\Pr(a|a) = 1/3, \Pr(b|a) = 1/3, \Pr(c|a) = 1/3 \quad \text{Codice: } \begin{matrix} a \to 0 \\ b \to 10 \\ c \to 11 \end{matrix} \quad l^a_{avg} = 5/3 \text{ bit}$$

Stato b:

$$\Pr(a|b) = 1/4, \Pr(b|b) = 1/2, \Pr(c|b) = 1/4 \quad \text{Codice: } \begin{matrix} a \to 10 \\ b \to 0 \\ c \to 11 \end{matrix} \quad l^b_{avg} = 3/2 \text{ bit}$$

Stato c:

$$\Pr(a|c) = 1/4, \Pr(b|a) = 1/4, \Pr(c|a) = 1/2 \quad \text{Codice: } \begin{matrix} a \to 10 \\ b \to 11 \\ c \to 0 \end{matrix} \quad l^a_{avg} = 3/2 \text{ bit}$$

The total average length can be calculated as the average of the conditional average lengths using the asymptotic probabilities of the states, i.e.

$$l_{avg} = p_a l^a_{avg} + p_b l^b_{avg} + p_c l^c_{avg} = 17/11 \cong 1.5454 \; bit \tag{3.66}$$

Let's compare this length with the minimum length (i.e., entropy) obtainable from a memoryless source S_{sm} having symbol probabilities equal to the asymptotic probabilities of the source with a given memory (the so-called *adjunct* source):

$$H(S_{sm}) = p_a \log\left(\frac{1}{p_a}\right) + p_b \log\left(\frac{1}{p_b}\right) + p_c \log\left(\frac{1}{p_c}\right) \cong 1.5726 \tag{3.67}$$

Such entropy is *greater* than the average length of our simple encoding scheme! As we already knew, the conclusion is that sources with memory can be in general compressed more than memoryless ones.

Computing the entropy rate of the Markov source is simple. The entropy of S_{cm} conditioned on a particular state is easily calculated as the average of the conditioned information:

$$H(S_{cm}|s_k) = \sum_{m=1}^{M} p(s_m|s_k) \text{In}(s_m|s_k) = \sum_{m=1}^{M} p(s_m|s_k) \log\left(\frac{1}{p(s_m|s_k)}\right) \tag{3.68}$$

The entropy rate is obtained as the average on the asymptotic probabilities of the conditional entropies of the different states:

$$H(S_{cm}) = \sum_{k=1}^{M} p(s_k) H(S_{cm}|s_k) = \sum_{k=1}^{M} p(s_k) \sum_{m=1}^{M} p(s_m|s_k) \log\left(\frac{1}{p(s_m|s_k)}\right)$$

$$= \sum_{k=1}^{M} \sum_{m=1}^{M} p(s_k) p(s_m|s_k) \log\left(\frac{1}{p(s_m|s_k)}\right) = \sum_{k=1}^{M} \sum_{m=1}^{M} p(s_m, s_k) \log\left(\frac{1}{p(s_m|s_k)}\right) \tag{3.69}$$

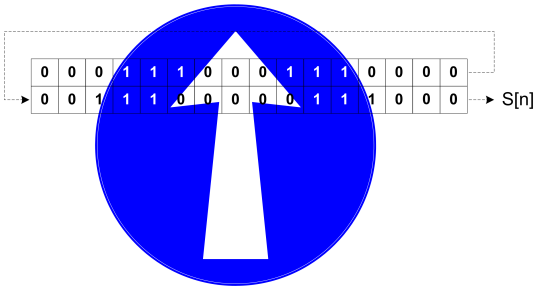


Figure 3.17 Scanning of a B/W image

In our example, we have

$$H(\Omega_m) = 1 + \tfrac{1}{3}\log(\tfrac{1}{3}) \cong 1.5283 \text{ bit/simbolo} \tag{3.70}$$

Our per-state Huffman coding comes very close to the theoretical minimum. It can be also shown that the entropy of the Markov source with memory is always less than that of the adjunct source.

Example 3.16

Coming back to the example of scanning an image, the reasder may remember that we modeled the scan of a printed text as a memoryless, unbalanced source, that is, with many more 0 symbols than 1 symbols: $p_1 = 0.9$, $p_2 = 0.1$. However, if we were to scan any black and white image (Fig. 3.17)at 1 binary symbol per pixel, the number of 1s and 0s this time would be in general well balanced. How to compress it then? As already observed, an image is generally source with memory, and in fact the string output of the scanner would be in this case

111111111111000000000000000000000000011111111111111100000000000000

that is, with long strings of 1s and 0s (areas of uniform color on the image), but with a globally similar number of 1s and 0s.

The correct modeling is now assuming a Markov source, with transition probabilities

$$p(0|0) = p(1|1) = 0.9 \ , \ \ p(0|1) = p(1|0) = 0.1 \tag{3.71}$$

This shows that the source has an inherent trend to stay for a long time into its own state, sporadically transitioning to the other. The source is also symmetric, that is, the strings of 1s and 0s are on average the same length (the transition matrix is symmetric).

Let us compute the asymptotic probabilities:

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \ , \ \ p_1 + p_2 = 1 \tag{3.72}$$

Taking the first equation and using $p_2 = 1 - p_1$ we get $0.9p_1 + 0.1(1 - p_1) = p_1$ from which $p_1 = p_2 = 0.5$, as expected.

The entropy rate is also

$$H_\infty(S) = p_1 H(S|0) + p_2 H(S|1) = 0.5\eta(0.1) + 0.5\eta(0.1) = \eta(0.1) \tag{3.73}$$

Although the marginal probabilities of 0 and 1 are 0.5, the source carries much less information than $H_{max} = 1$, as we already know.

There is no general optimal strategy for encoding sources with memory. One simple possibility is to consider the k-th extension of the source, provided, of course, that its statistical description is known, that is, the joint probabilities of order k of the new extended symbols are available. For a Markov chain source, this description is always computable via the transition matrix and the so-called *chain rule*. The new source can then be approximated as memoryless and can be encoded, for example, with a Huffman code. This procedure, as we have already seen in the case of memoryless sources, is asymptotically optimal (that is, optimal for $k \to \infty$), but involves an exponential increase in complexity.

Another general compression strategy (called Δ coding) follows the scheme of the DPCM encoder in the previous Section. If we can build, by exploiting the memory properties of the source, a good predictor $\hat{S}[n]$ of the current symbol $S[n]$, it is worth considering the difference signal or prediction error signal $E[n] = S[n] - \hat{S}[n]$. Considering binary sources, if the predictor has good characteristics, $E[n]$ will consist of long sequences of 0s interspersed with a few 1s here and there, and can therefore be efficiently encoded with a repetition-length code. The Δ strategy is particularly useful for sources with long memory, that is, those that cannot be modeled as simple Markov chains.

Example 3.17

Let's take back the source from Example 16 and try to compress it using the "Δ" strategy. Since the source tends to stay at one level, we simply use the previous value as a predictor for the next value: $\hat{S}[n] = S[n-1]$, adopting a trivial differential encoding. What are the statistics of $E[n] = S[n] - \hat{S}[n]$? We have

$$\Pr\{E[n] = 0\} = \Pr\{S[n] = S[n-1]\} =$$

$$= \Pr\{S[n] = 0 | S[n-1] = 0\}\, p_1 + \Pr\{S[n] = 1 | S[n-1] = 1\}\, p_2 = 0.9$$

therefore $E[n]$ is, as expected, unbalanced toward 0, giving rise to (more) easily compressible strings.

We'll look later into more source preprocessing strategies to make it easily compressible, such as the Burroughs-Wheeler transform.

3.7 Adaptive universal lossless encoding of sources with memory

All of the coding schemes presented so far are based on a priori knowledge of the source characteristics. The Huffman code for memoryless sources is based on knowledge of symbol probabilities, and the state-based code for Markovian sources also requires knowledge of transition probabilities. Sources with a larger memory than Markovian ones would even require knowledge of higher-order joint or conditional probabilities for optimal compression. Even the simple RLE algorithm must know symbol probabilities in order to establish how many bits to allocate to run lengths. Conversely, in practical applications, the statistics of the source are often unknown or vary over time. Therefore, a coding scheme is needed that

can work with any source, i.e., is *universal*. Since such coding is able to adapt to sources with various characteristics, we could call it (but the term is less common in this context) *adaptive*. One of the most widespread compression techniques is based on so-called LZ (i.e., Lempel-Ziv, the inventors' surnames) algorithm from the late 1970s. Various versions of LZ are used in practice. The lossless coding standard for representing images in the GIF format is based on this algorithm, and the compressed storage format of `.zip` files for the Windows operating system is also based on a variant of the LZ coding called LZ77, which is very similar to LZ. The `.zip` algorithm (developed by P. Katz in the mid-1980s) is based on LZ77 followed by Shannon-Fano encoding, and to implement the latter it uses an estimate of the symbol probabilities derived as a by-product of the LZ77 itself.

The Lempel-Ziv code and its modifications base their operation on a code table that maps variable-length source words (so-called *phrases*) to fixed-length codewords – this is the opposite of the Huffman code. The mapping is done so that "long" and "frequent" source phrases are mapped to a shorter codeword. This lossless compression strategy is particularly effective in sources that contain sequences that recur unchanged, even if they are repeated irregularly. In this strategy, two main problems arise: i) how to construct the table, and ii) how to communicate the table to the decoder. The solution is simple and effective: i) the table is constructed adaptively real-time by appropriately segmenting the source into phrases, and ii) the table can be reconstructed by the decoder simply by observing the encoded string.

Let us make a simple example to understand how LZ works, assuming we use a table of 16 4-bit codewords. The table starts with the two initial codewords 0000 and 0001, corresponding to the source symbols 0 and 1.

Assume that the sequence to be encoded is (no wonder, it's a sequence like any other...)

0101010101010101010101010101010101010101010101010101010101010101010101...

The encoder begins scanning the sequence to be encoded to parse it, looking for the shortest sentence not yet stored in the table. Once found, that phrase is stored into the table and assigned the first word in the table that is not yet assigned as a codeword (for us, on 4 bits). In the example, the first sentence (subsequence) is 01, and therefore the first encoding step is:

01,0101010101010101010101010101010101010101010101010101010101010101010101...

0 :	0000
1 :	0001
01 :	**0010**

The new phrase is *not* encoded with the (new) word just entered into the table, but as the concatenation of the encoding of the word already present followed by the encoding of the last character (bit) of the new sentence: 0000**0001**. This is the codeword that is produced. This process continues until the table is filled up, so that the parsed input string, the table, and the codewords are as follows:

01,010,10,101,0101,01010,1010,10101,010101,0101010,101010,1010101,01010101,010101010, ...

0 :	0000	1010 :	1000
1 :	0001	10101 :	1001
01 :	0010	010101 :	1010
010 :	0011	0101010 :	1011
10 :	0100	101010 :	1100
101 :	0101	1010101 :	1101
0101 :	0110	01010101 :	1110
01010 :	0111	010101010 :	1111

0000**0001**,0010**0000**,0011**0000**,0100**0001**,0011**0001**, 0110**0000**,0101**0000**,1000**0001**,0111**0001**, 1010**0000**,1001**0000**,1100**0001**,1011**0001**,1110**0**

From now on, the encoding proceeds in a purely tabular fashion, using the now fixed and no longer adaptive table.

The reason why the first codeword is not that of the newly discovered sentence, but the concatenation of a pre-existing word and a new character, stems from the decoder's operation. The decoder begins decoding with the sole initialization words 0000 and 0001, corresponding to 0 and 1, and therefore could not decode the new word. It can only rely on words already present, so that it can immediately decode the first phrase, namely 01. Realizing that 01 is not in the table, the decoder stores it down, and from there on it closely follows the evolution of the coding table step by step.

Sooner or later, the source segmentation discovers long sequences that are typical of the source, which are encoded with shorter words and make up for the core of compression. In this sense, the code is adaptive and automatically adjusts to the statistical characteristics of the source, including memory. During encoding, the compression ratio will be good because the last entries in the table will be used primarily, for which the code word is (much) shorter than the source one. LZ77 monitors the compression ratio in real time: if by chance the ratio tends to degrade, the table is reset and encoding starts over again. This can happen, for example, when a file to be compressed first contains text, with certain statistical properties, followed by an image of a figure, bearing different characteristics, for which the table used up to that point is not optimal.

In commercial implementations (the `.zip` format for Windows or the `compress` algorithm for Linux), groups of 8 consecutive bits (bytes) are considered as elementary source symbols, and the table is therefore initially filled with the 256 possible values of these symbols, from 00000000 to 11111111. Code words are 12-bit long for a dictionary of 4096 code symbols, of which the first 256 are the elementary source symbols with 4 leading zeros.

3.8 Source compression and cata compression: The Burrows-Wheeler transform

Classical algorithms for source compression have evolved into an endless number of more modern, efficient technologies for data compression that are applied in a multitude of ICT fields. Such variants have in turn spawned further technologies and algorithms for data parsing, efficient data representation/storage, etc., which are admittedly computer-science related and are beyond the scope of these notes. However, we wish to describe

one of the ancestors of these algorithms, which is not strictly speaking a compressor, but a preprocessing technology that facilitates the task of subsequent true (compression, parsing, storage) algorithms that may be necessary: the Burrows-Wheeler transform (BWT), which, as we will see, is very effective at preprocessing sources with memory, particularly those characterized by words (i.e., long strings of source symbols) that frequently recur unchanged.

The BWT is applied to a (finite) string of source symbols of length N and produces another string of source symbols of length N obtained from a *permutation* of the original string. The permutation, as such, is invertible, and therefore the original string can be reconstructed from the transform. Suppose our source has the alphabet $\Omega = \{a, h, i\}$ and that the string needs to be preprocessed is

```
ahiahiahiahiahi
```

We append to the string an end-of-file (EOF) symbol that does not belong to Ω, such as #, which we must interpret as preceding the first symbol $s_0 = a$ in the alphabet (like a "s_{-1}"). Now we imagine forming a symbol matrix of size $(N + 1) \times (N + 1)$ obtained by placing the given string (including the final #) in the first row, and by inserting in the subsequent rows all of the cyclic left shifts of one position from the previous row. We say "imagine" because the matrix is not actually computed by BWT implementations – the operation is conceptual. We obtain the following, left-hand side table:

ahiahiahiahiahi#	#ahiahiahiahiahi
hiahiahiahiahi#a	ahi#ahiahiahiahi
iahiahiahiahi#ah	ahiahi#ahiahiahi
ahiahiahiahi#ahi	ahiahiahi#ahiahi
hiahiahiahi#ahia	ahiahiahiahi#ahi
iahiahiahi#ahiah	ahiahiahiahiahi#
ahiahiahi#ahiahi	hi#ahiahiahiahia
hiahiahi#ahiahia	hiahi#ahiahiahia
iahiahi#ahiahiah	hiahiahiahiahi#a
ahiahi#ahiahiahi	hiahiahi#ahiahia
hiahi#ahiahiahia	hiahiahiahi#ahia
iahi#ahiahiahiah	iahiahiahiahi#ah
ahi#ahiahiahiahi	i#ahiahiahiahiah
hi#ahiahiahiahia	iahi#ahiahiahiah
i#ahiahiahiahiah	iahiahi#ahiahiah
#ahiahiahiahiahi	iahiahiahi#ahiah

Now now sort the matrix by rows, considering the rows as source words and sorting them in lexicographical (alphabetical) order, also taking into account #, obtaining the table on the right-hand side above. The first row contains the original string preceded by #, and the first column contains all the characters of the string arranged in alphabetical order. The transformed string is the string obtained by reading the *last column* of the reordered matrix, removing the # escape symbol, but adding the number of the raw where it was found:

```
iiiiihhhhhaaaaa5
```

Now we show that the BWT is invertible. Before doing this, lat us comment on the utility of the output: the source recurrences have been identified and grouped, so that a simple compression algorithm like RLE can be applied. The overall procedure turns out to be a very simple yet highly effective data compression algorithm.

To invert the BWT, we start by reconstructing the last column of the reordered matrix (adding # in position 5), and, after that, immediately reconstructing the first column, that we can obtain by alphabetically rearranging all the characters of the last column:

```
#xxxxxxxxxxxxxxi      i# #a      #axxxxxxxxxxxxxi
axxxxxxxxxxxxxxi      ia ah      ahxxxxxxxxxxxxxi
axxxxxxxxxxxxxxi      ia ah      ahxxxxxxxxxxxxxi
axxxxxxxxxxxxxxi      ia ah      ahxxxxxxxxxxxxxi
axxxxxxxxxxxxxxi      ia ah      ahxxxxxxxxxxxxxi
axxxxxxxxxxxxxx#      #a ah      ahxxxxxxxxxxxxx#
hxxxxxxxxxxxxxxa      ah hi      hixxxxxxxxxxxxxa
hxxxxxxxxxxxxxxa      ah hi      hixxxxxxxxxxxxxa
hxxxxxxxxxxxxxxa      ah hi      hixxxxxxxxxxxxxa
hxxxxxxxxxxxxxxa      ah hi      hixxxxxxxxxxxxxa
hxxxxxxxxxxxxxxa      ah hi      hixxxxxxxxxxxxxa
ixxxxxxxxxxxxxxh      hi i#      iaxxxxxxxxxxxxxh
ixxxxxxxxxxxxxxh      hi ia      iaxxxxxxxxxxxxxh
ixxxxxxxxxxxxxxh      hi ia      iaxxxxxxxxxxxxxh
ixxxxxxxxxxxxxxh      hi ia      iaxxxxxxxxxxxxxh
ixxxxxxxxxxxxxxh      hi ia      iaxxxxxxxxxxxxxh
```

Now, I have all the character pairs from the source, considering that the rows are cyclic. By re-ordering alphabetically the pairs, I obtain the first two columns, and so on. At the end, the first row, after removing the first character, will be the original string. The forward and inverse transform algorithms do not require explicit matrix construction: the various sequences are obtained by cyclically playing with the indices of the string – both algorithms are of *linear* complexity with N, making the BWT very efficient.

As already stated, the BWT is just a preprocessing algorithm: it does not compress data per se, but prepares the source for simple compression – it is used in various compression softwares and as a preprocessor for DNA sequencing in bioinformatics. If anything, BWT highlights the great gap between the clear and powerful conceptual framework of Information Theory and the clever implementations that come from an empirical yet effective approach.

3.9 Lossy source coding

All of the source encodings algorithms examined so far are lossless and in most cases guarantee significant but not dramatic compression ratios (around 50% for a text file). To get to higher compression ratios (tens to some hundreds), lossy source compression strategies are widely used. Many lossy source compression techniques are based, as we will see, on a series of empirical criteria that are highly dependent on the application. Is there a formal framework in which these techniques can be combined and formalized? We will address this question in Chapter **??** and discover that the answer is positive. Before examining that framework, we need to make a long digression about reliable transmission of information over noisy channels – this topic, relevant by itself, is in fact connected to the issue of lossy coding.

The basic scheme for reliable transmission over a noisy channel is shown in Fig. 3.18, in which the source information gets to the user partially distorted. This distortion (in the non-technical sense) of information can be due to uncontrollable and undesirable

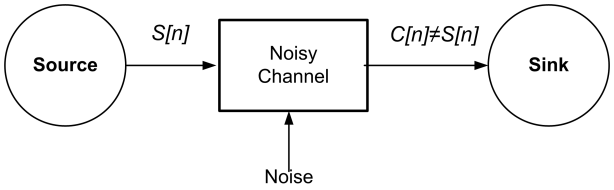
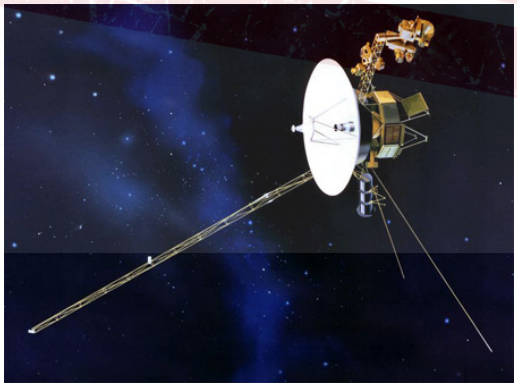


Figure 3.18 Comunicazione su di un collegamento con disturbo

external perturbations (noise, interference, etc.), as is the case of wireless communications, for example. Such a scheme is also applicable to the case wherein the "channel" is an encoding/decoding process that intentionally distorts the source information in order to reduce its rate. The study of the problem of reliable transmission over a noisy channel, which we will address in the next section, allows us to introduce concepts that will also be the basis of the problem of lossy source compression, which will be addressed in Chapter **??**.

CHAPTER 4

RIDERS OF THE LOST RELIABILITY



"Artist's impression of the Voyager 1 satellite, which was launched in 1977 and is still transmitting solar system exploration data."

—Communications against all odds

The Voyager 1 satellite is capable of reliably transmitting digital data at 140 to 1,400 bits/s over the greatest distance ever traveled by a human-made object: currently (2025) it has passed the boundaries of the solar system and is about 168 astronomical units ($25 \cdot 10^9$

km) from Earth. The motto of this chapter is, in fact: communicating in spite of all troubles - that is, ensuring the reliability of the data received over a link despite various sources of interference and noise.

4.1 Digital communications over a "noisy channel"

In the previous paragraphs, we established how to characterize the information content of an information source, and we examined some techniques to make a source as non-redundant as possible, with or without loss of fidelity. That is, we looked at how to make the communication/storage link *efficient*, using the minimum amount of data possible.

That said, we remind the reader that the purpose of a communication system is to transfer a certain amount of information from a source $S[n]$ to a user in a certain time interval. Often, this transmission occurs in *real time* so that the replica of the sequence of symbols received by the user is as synchronous as possible with the source. This is the case with the transmission of voice signals for a telephone call, or video signals in a videoconference. Other times, communications may occur with a (large) delay or in *virtual time*, as in the case of transferring a data file (for example, a banking transaction) between two servers, where synchrony is completely irrelevant, or in the case of broadcasting of recorded material (films, shows, etc.). In any case, depending on the type of source, the system must guarantee a certain quality of service (QoS), usually expressed in terms of signal-to-noise ratio, bit error rate (BER), maximum delay, probability of packet loss, and so on. In general, when considering the physical medium across which communications takes place, it is not possible to transfer information unaltered because of noise, interference, imperfections, or limitations of the physical devices and communication medium (radio, fiber, etc.).

A similar problem is encountered when storing and retrieving the source message on a medium of limited capacity. The problem in this case is constituted by write/read errors on the storage medium (typically a hard disk) that can occasionally occur. In the process of read/write, we have the same kind of information loss that we encounter when communicating over a noisy channel. Such a loss becomes larger and larger (so that reliability is smaller and smaller) if we try to squeeze many information bits onto a limited physical medium, for example by making the magnetic pits of the HD smaller and smaller – there may be a relation between the desired reliability and the achieved capacity of the medium.

We have outlined the two fundamental problems of information processing (transmission/storage), namely, reliability, and capacity. In the following paragraphs, we will examine the interrelationship between these two aspects, and the strategies that can be adopted to optimally exploit a given channel.

4.2 Il canale di comunicazione nella Teoria dell'Informazione

4.2.1 Modelli di Canale

To visualize the meaning of "quality of service", we show in Fig. 4.1 the test image "Lena" as produced by the source (a) and in two cases of transmission with loss of information. In the first case (b), the bits constituting the information are randomly flipped with probability 10^{-4} (BER, Bit Error Rate), while in the second (c) the flipping probability is $5 \cdot 10^{-4}$. The alteration of the source is evident. If such probability is too high, some form of

Figure 4.1 Original source (a), intermediate BER (b), high BER (c)

countermeasure must be adopted to make the representation of the source as *robust* as possible.

As usual, we start with an example from basic wireless communications: a digital link via 16-QAM modulation on an AWGN (Additive White Gaussian Noise) channel, as shown in Fig. 4.2 (a). This schematic describes in detail the physical layer of the digital link, but if we want to build slightly more abstract models of this link, we have various options, further shown in Fig. 4.2: ignoring the actual structure of what is enclosed within the dashed block, in Fig. 4.2 (b) we have a channel with binary input and binary output. On the contrary, in Fig. 4.2 (c) we have a channel with 16 input and 16 output symbols, and finally in 4.2 (d) we have binary input and "continuous" output (also called *soft* as opposed to the *hard* output produced by the threshold device). In the following, will try to analyze abstract models that do not depend on the actual implementation of the physical layer, and only consider the kind of processing of information at the input/output terminals, i.e., *end-to-end* models.

Let's start with a simpler case than the one just examined, specifically, a BPSK transmission with AWGN noise. From the fundamentals of telecommunications, we can easily derive the error probability $P(E)$ on the generic source bit, calculating as an intermediate step the error probabilities *conditional* on the two possible source values, $P(E|0)$ and $P(E|1)$:

$$P(E|0) = P(\hat{S}[n] = \hat{1}|S[n] = 0) \stackrel{\triangle}{=} P(\hat{1}|0) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

$$P(E|1) = P(\hat{S}[n] = \hat{0}|S[n] = 1) \stackrel{\triangle}{=} P(\hat{0}|1) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{4.1}$$

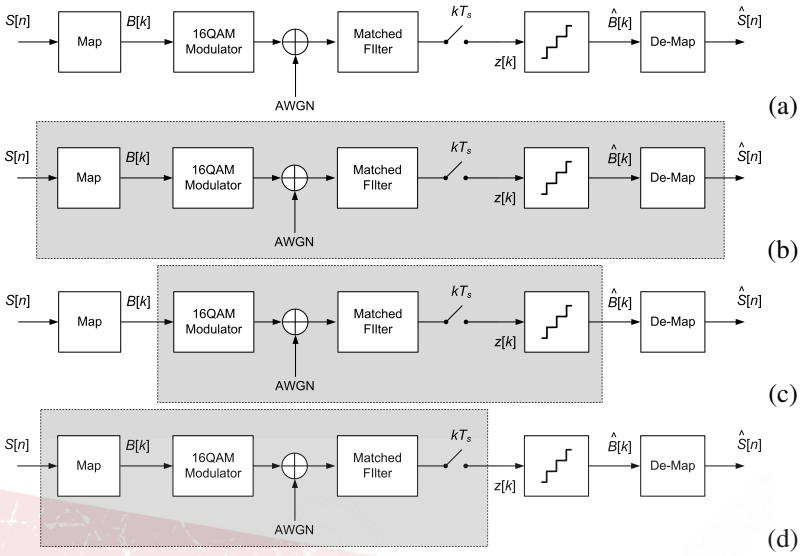


Figure 4.2 Diversi modelli del link digitale 16-QAM con rumore

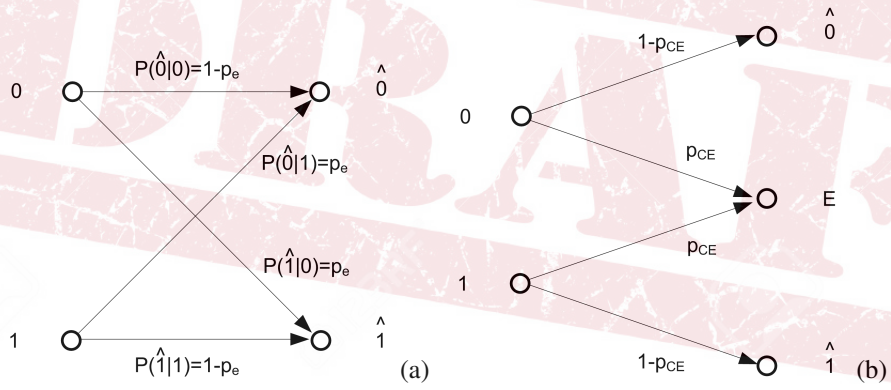


Figure 4.3 Representation of two noisy channels: BSC BSC (a), and BEC (b)

where $\hat{S}[n]$ is the reconstructed (estimated) version at the receiver of the transmitted source symbol $S[n]$, and where we see the symmetry of the problem induced by the equality of the two conditional probabilities.

Based on our knowledge of these quantities, and following the *modus operandi* just outlined, we can forget about the actual physical structure of the system (modulation, noise level, etc.) and move to a more abstract model of a noisy channel, which we represent graphically in Fig. 4.3 (a). The meaning of this representation is clear: if the source generates a symbol "0", a symbol "$\hat{1}$" will be received (erroneously) with probability $P(\hat{1}|0)$, and a symbol "$\hat{0}$" will be received (correctly) with probability $P(\hat{0}|0) = 1 - P(\hat{1}|0)$. Similarly, for the transmitted symbol "1". Characterizing the channel therefore requires knowledge of the four conditional probabilities. $P(\hat{S}[n] = s_m|S[n] = s_k) \stackrel{\triangle}{=} P(s_m|s_k)$, $m, k = 1, 2$. In general, the channel output can have an alphabet with a different number of symbols than the source, when we want to model situations other than the one just seen.

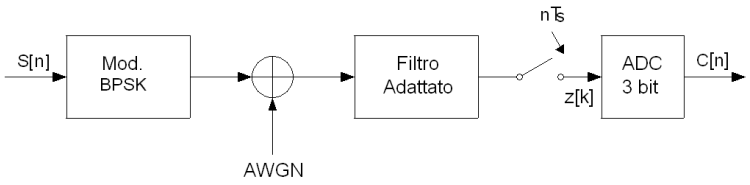


Figure 4.4 BPSK link with 3-bit quantization of the output signal

As a second example, we consider sending a data packet into an IP network, disregarding physical layer technologies altogether. At any node in the network, before forwarding the packet to the next node or to the end user, the packet's integrity is checked for possible communication errors by performing a parity check on the appropriate CRC (Cyclic Redundancy Check) field in the preamble. If the parity check fails, the packet is simply discarded and removed from the network – this is called a packet frop or *packet erasure*. If, however, the parity check is successful, the packet is forwarded or used because it is error-free.

This situation is represented in the diagram in Fig. 4.3 (b), where it is noted that probabilities of the kind $P(\hat{1}|0)$ or $P(\hat{0}|1)$ do not appear because they are considered equal to 0 - the channel does not allow for error events. Instead, a third outcome of the bit transmission appears explicitly, namely the deletion of the bit because it belongs to a packet discarded by the parity check - whether it was among those of the packet received correctly or among those that caused the check to fail, does not matter. The Binary Erasure Channel (BEC) model is shown in Fig. 4.3 (b), where $P_{CE} \stackrel{\triangle}{=} P(CE|0) = P(CE|1)$ is the probability of a *Channel Erasure* at level 0 or 1, respectively.

Another example of a channel with different input and output alphabets is that of a BPSK modem, in which the received analog signal is digitized (i.e., sampled and quantized) to 3 bits, as shown in Fig. 4.4. Then the output of channel $C[n]$ has an alphabet $\Theta = \{c_1, c_2, ..., c_8\}$ made of 8 symbols (the eight possible values of the three bits).

Generalizing, a memoryless discrete-noise communication channel is characterized by the $M \times L$ matrix of conditional probabilities

$$\mathbf{P} = \{P(c_k|s_m)\}\ ,\ k = 1, ..., K\ ;\ m = 1, ..., M \tag{4.2}$$

which are also called *channel probabilities*. The classic AWGN channel with multi-level modulation is a hard-decision "square" channel for which $K = M$ (as is the case in Fig. 4.2 (b)). In this case, the total error probability is easily calculated:

$$P(E) = \sum_{m=1}^{M} P(E|s_m)P(s_m) = \sum_{m=1}^{M} [1 - P(c_m|s_m)]P(s_m) \tag{4.3}$$

where the a priori probabilities $P(s_m)$ are known from the source description. The reader should explain how the matrix $\mathbf{P}$ looks like for a square channel free of errors, and how it looks for a channel with noise outweighing the signal. The example associated with BPSK transmission on a channel with AWGN in which the two conditional error probabilities are equal, is modeled by a BSC.

4.2.2 Equivocation, Mutual Information, Irrelevance

Coming back to the general case, the probabilities associated with the source symbols s_m determine the information content of the source, $H(S[n]) = -\sum_{m=1}^{M} P(s_m)\log(P(s_m))$.

Starting from the column vector $\mathbf{p_s}$ of source probabilities (or input probabilities, known to the receiver), we can easily calculate the column vector $\mathbf{p}_c$ of channel (or output) symbol probabilities via the channel probability matrix:

$$\mathbf{p}_c = \mathbf{P}\mathbf{p} \tag{4.4}$$

The matrix $\mathbf{P}$ in fact has the meaning of a transition matrix. This new probability distribution in turn determines the information content of the symbols $C[n]$ at the channel output:

$$H(C) = \sum_{k=1}^{K} P(c_k) \log\left(\frac{1}{P(c_k)}\right) \tag{4.5}$$

Making (4.4 explicit, we note that it is essentially a vector formulation of the total probability theorem: , $P(c_k) = \sum_{m=1}^{M} P(c_k|s_m)P(s_m)$, $k = 1, ..., K$.

What is the relationship between the two informations (entropies) $H(S)$ and $H(C)$? What is the effect of transmission on the noisy channel on the information content of the source? To answer this question, we will adopt the point of view of the receiver who knows the a priori probabilities of the source but who, having observed a certain value of $C[n]$, is not entirely certain (due to the noisy nature of the channel) about the value of $S[n]$ that was actually transmitted: there is a residual uncertainty in $S[n]$ even after the channel output has been observed. Assuming that the channel value c_k has been observed, this conditional uncertainty is the entropy of the source conditioned on the particular value c_k:

$$H(S|c_k) = \sum_{m=1}^{M} P(s_m|c_k) \log\left(\frac{1}{P(s_m|c_k)}\right) \tag{4.6}$$

Considering all possible channel symbols and averaging them, we obtain the definition of average output uncertainty, or *equivocation*:

$$\begin{aligned} H(S|C) &= \sum_{k=1}^{K} P(c_k) \sum_{m=1}^{M} P(s_m|c_k) \log\left(\frac{1}{P(s_m|c_k)}\right) \\ &= \sum_{k=1}^{K} \sum_{m=1}^{M} P(s_m, c_k) \log\left(\frac{1}{P(s_m|c_k)}\right) \end{aligned} \tag{4.7}$$

Before observing the channel output, the receiver's uncertainty about the transmitted symbol is equal to the "a priori" uncertainty, that is, $H(S)$. *After* observing the channel output, the residual uncertainty is $H(S|C)$. For example, if the channel is noiseless, all the uncertainties $H(S|c_m)$ are zero, and so is $H(S|C)$: the uncertainty about the source symbols is zero because, with no noise, there is no possibility of equivocation – this is called the *ideal* channel. If, in the limit, the channel is maximally disturbed, $C[n]$ is completely independent of $S[n]$, the probabilities of the source symbols are not affected by the conditioning to (i.e., observation of) the channel outputs, therefore $H(S|C) = H(S)$ and the equivocation is maximum – this is called the *useless* channel !

The difference between the source information and the channel equivocation $H(S) - H(S|C)$ (which is $H(S)$ for a noiseless channel or 0 for a "noise-only" channel) has an important significance. It is called *mutual information* and represents the amount of information about the source that the channel is actually able to convey despite the presence of noise:

$$I(S, C) \triangleq H(S) - H(S|C) \tag{4.8}$$

We will show that $I(S,C) \geq 0$, but since $H(S|C) \geq 0$ (average of entropies), we see that the amount of information conveyed by the channel is always *less* than the source information due to the presence of the noise, as one might expect. Let's now work out the definition of mutual information:

$$I(S,C) = \sum_{m=1}^{M} P(s_m) \log\left(\frac{1}{P(s_m)}\right) + \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k)\log(P(s_m|c_k))$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k) \log\left(\frac{1}{P(s_m)}\right) + \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k)\log(P(s_m|c_k))$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k) \log\left(\frac{P(s_m|c_k)}{P(s_m)}\right)$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k) \log\left(\frac{P(s_m,c_k)}{P(s_m)P(c_k)}\right) = I(C,S) \tag{4.9}$$

From (4.9) we note the formal symmetry in $S[n]$ and $C[n]$ of the definition of $I(S,C) = I(C,S)$, so we can also write:

$$I(S,C) = H(S) - H(S|C) = H(C) - H(C|S) = I(C,S) \tag{4.10}$$

From this we obtain a balance equation between the input and the output, represented in Fig. 4.5:

$$H(C) = H(S) - H(S|C) + H(C|S) = I(S,C) + H(C|S) \tag{4.11}$$

The interpretation of (4.11) is interesting: the information content of the channel output is made up of a useful part $I(S,C)$ given by the amount of source information coneyed by the channel, plus a second irrelevant component $H(C|S)$ (called *channel irrelevance* or irrelevance *tout-court*) which is information itself, but coming from another source external to the S, that is, generated by noise. If, in the limit, the channel is noise-only, the mutual information is zero, the irrelevance is maximum, the channel (output) entropy is only made of irrelevance, and it can even be greater than that of the source (think of channel symbols uniformly distributed due to a large noise overcoming the useful signal).

We are now to examine some formal properties of mutual information in relation to the joint entropy of S and C, defined as follows:

$$H(S,C) \stackrel{\triangle}{=} \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m,c_k) \log\left(\frac{1}{P(s_m,c_k)}\right) \tag{4.12}$$

We will not dwell on these properties, which can be easily derived by the reader as an exercise: for example, find

$$I(S,C) = H(S) + H(C) - H(S,C) \tag{4.13}$$

e give the proper interpretation. In addition, exploiting Gibb's inequality (3.20), show that $I(S,C) \geq 0$ and find when $I(S,C) = 0$.

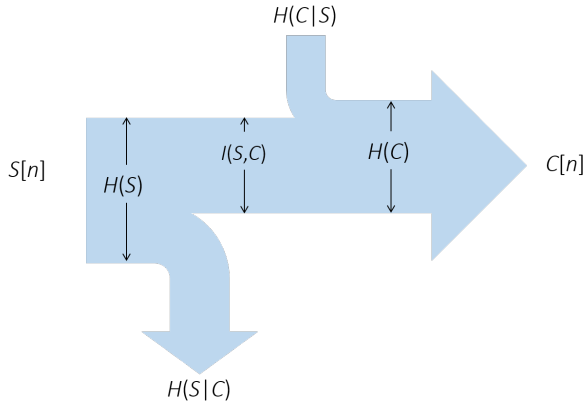


Figure 4.5 Information flow on the noisy channel

4.2.3 Special channels

Let us compute the mutual information of the BSC with transition probability $P(E)$. Denoting p the a priori probability of the symbol 0, $p \stackrel{\triangle}{=} P(0)$ we have:

$$I(S,C) = H(S) - H(S|C) = H(C) - H(C|S) \tag{4.14}$$

Concentrating on $H(C|S)$ we have

$$H(C|S) = H(C|0)P(0) + H(C|1)P(1) = H(C|0)p + H(C[n])(1-p) \tag{4.15}$$

The conditional entropies mentioned above are simple to evaluate:, once 0 is the input symbol, the output will be $\hat{1}$ 1 with probability $P(E)$ and $\hat{0}$ with probability $1 - P(E)$. This conditional entropy is therefore that of a binary source with probability of a symbol equal to $P(E)$, which we defined in (3.18) as $\eta\left(P(E)\right)$. The same reasoning, applies when conditioning the source symbol to 1 for symmetry, so that

$$H(C|S) = \eta\left(P(E)\right)P(0) + \eta\left(P(E)\right)P(1) = \eta\left(P(E)\right) \tag{4.16}$$

that does not depend on the source probabilities. In addition, the output of the channel is binary, so that $H(C) = \eta\left(P(\hat{0})\right) = \eta\left(P(\hat{1})\right)$. Using the total probability theorem,

$$P(\hat{0}) = (1 - P(E)) \cdot p + P(E) \cdot (1-p) \tag{4.17}$$

therefore

$$I(S,C) = \eta\left([1 - P(E)] \cdot p + P(E) \cdot (1-p)\right) - \eta\left(P(E)\right) \tag{4.18}$$

Equally simple is the calculation of mutual information for the BEC with source probabilities p and $1 - p$:

$$I(S,C) = H(S) - H(S|C) = \eta\left(p\right) - H(S|C) \tag{4.19}$$

Concerning $H(S|C)$:

$$H(S|C) = H(S|\hat{0})P(\hat{0}) + H(S|E)P(E) + H(S|\hat{1})P(\hat{1}) \tag{4.20}$$

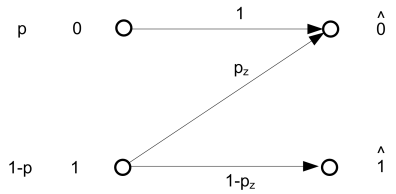


Figure 4.6 Z channel

We see that $H(S|\hat{0}) = H(S|\hat{1}) = 0$ because whenever $\hat{0}$ or $\hat{1}$ is received, there is no uncertainty about the symbol sent by the source (which are certainly 0 or 1, respectively), therefore the (conditional) entropy is equal to 0. When E is received, the uncertainty is that of a binary source with probability

$$P(0|E) = \frac{P(E|0)P(0)}{P(E)} \tag{4.21}$$

using Bayes' rule. On the other hand, by total probability theorem,

$$P(E) = P(E|0)P(0) + P(E|1)P(1) = p_{CE}p + p_{CE}(1-p) = p_{CE} \tag{4.22}$$

so that $P(0|E) = p_{CE} \cdot p/p_{CE} = p$, $H(S|E) = \eta\,(p)$, and putting all together,

$$I(S,C) = H(S) - H(S|C) = \eta\,(p) - p_{CE}H(S|C) \cdot \eta\,(p) = (1-p_{CE})\eta\,(p) \tag{4.23}$$

The best-known example of an asymmetric channel is the so-called Z-channel, whose transition diagram is shown in Fig. 4.6. A situation of this type occurs in fiber-optic communications, where (in the shot-noise regime) an error on symbol 1 is much more likely than on level 0, and therefore in our model $P(\hat{1}|0)$ is approximated to zero. The quantity p_z represents the probability of an error on symbol 1, so the total error probability is

$$P(E) = p_z(1-p) \tag{4.24}$$

Lat us find the mutual information of the Z channel:

$$I(S,C) = H(C) - H(C|S) \tag{4.25}$$

Since C is binary on $\hat{0}$ and $\hat{1}$, we know that $H(C) = \eta\left(P(\hat{0})\right)$. On the other hand

$$P(\hat{0}) = p \cdot P(\hat{0}|0) + (1-p) \cdot P(\hat{0}|1) = p + p_z(1-p) \tag{4.26}$$

Coming to irrelevance,

$$H(C|S) = p \cdot H(C|0) + (1-p) \cdot H(C|1) = (1-p) \cdot H(C|1) = (1-p)\eta(p_z) \tag{4.27}$$

In fact, conditioning on the source symbol 0, there is no uncertainty about the value of the channel output, therefore $H(C|0) = 0$. Conditioning on the level 1, on the contrary, the output is binary with probability $p_z, 1-p_z$. In summary,

$$I(S,C) = \eta\,(p + p_z(1-p)) - (1-p)\eta(p_z) \tag{4.28}$$

4.3 Shannon capacity of a noisy channel

From the standpoint of our channel modeling, assigning a transmission channel means assigning the (transition) matrix of channel probabilities $\mathbf{P} = \{P(c_k|s_m)\}\ ,\ k = 1, ..., K\ ;\ m = 1, ..., M$. Coming back to the expression of mutual information $I(S, C)$, we can write

$$I(S,C) = \sum_{k=1}^{K}\sum_{m=1}^{M} P(s_m, c_k)\log\left(\frac{P(s_m, c_k)}{P(s_m)P(c_k)}\right)$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{M} P(c_k|s_m)P(s_m)\log\left(\frac{P(c_k|s_m)}{P(c_k)}\right)$$

$$= \sum_{k=1}^{K}\sum_{m=1}^{M} P(c_k|s_m)P(s_m)\log\left(\frac{P(c_k|s_m)}{\sum\limits_{i=1}^{M} P(c_k|s_i)P(s_i)}\right) \tag{4.29}$$

We see that our mutual information depends on the channel through the transition probabilities (which are known), and also on the probability distribution $P(s_m)$ of the source. Therefore, given a certain channel, $I(S, C)$ may be greater or smaller depending on the characteristics of the source.

This observation allows us to give an important definition, namely the channel capacity c or *Shannon capacity* of the channel. This quantity is equal to the maximum mutual information that can be conveyed by the channel, depending on the source characteristics (probability distribution):

$$c \stackrel{\triangle}{=} \max_{\mathbf{p}} I(S,C) = \max_{\mathbf{p}} \sum_{k=1}^{K}\sum_{m=1}^{M} P(c_k|s_m)P(s_m)\log\left(\frac{P(c_k|s_m)}{\sum\limits_{m=1}^{M} P(c_k|s_m)P(s_m)}\right) \tag{4.30}$$

where, as already seen, $\mathbf{p} \stackrel{\triangle}{=} [P(s_1), P(s_2), ..., P(s_M)]^T$. If a given channel is to be used in a maximally efficient way, the source must in some sense be "matched" to the channel so as to exploit it at best. Maximizing c with respect to $\mathbf{p}$ can in general be achieved with the Lagrange multiplier method, under the *constraint* is $\sum_{m=1}^{M} P(s_m) = 1$ and assuming of course $P(s_m) \geq 0 \forall m$. In the next section we'll see some elementary examples for specific cases.

4.3.1 Shannon capacity of relevant channels

To derive the Shannon capacity of the BSC, we must maximize the mutual information with respect to p (4.18):

$$c_{BSC} = \max_{p} \{\eta\left((1 - P(E))\cdot + P(E)\cdot(1-p)\right) - \eta\left(P(E)\right)\} =$$

$$\max_{p} \{\eta\left((1 - P(E))\cdot + P(E)\cdot(1-p)\right)\} - \eta\left(P(E)\right) \tag{4.31}$$

Since the irrelevance does not depend on source probabilities, our mutual information is maximum when $H(C)$ is maximum, that is, when the channel output symbols are

equiprobable. In conclusion, the capacity of the symmetric binary channel with error probability $P(E)$ appears to be

$$c_{BSC} = 1 - \eta\left(P(E)\right) \quad \text{bit/simbolo} \tag{4.32}$$

We say *appears* because we need to verify that this $\max$ is actually achieved - in other words, what is the source probability law that allows us to make the two output symbol equiprobable ($P(C[n] = 0) = P(C[n] = 1) = 1/2$) so as to attain this capacity? As we have already shown, this is achieved if and only if $p = 1/2$, that is, with an equiprobable source - a result of channel symmetry.

Studying the (4.32), we note that by increasing the $P(E)$ from 0 to 1/2 (from noise-free to noise-only), channel capacity progressively decreases from 1 to 0. The reader should i) explain why, by increasing the $P(E)$ beyond 1/2, the capacity tends to increase again, and ii) try to generalize the concept of a symmetric channel also to the non-binary square case.

Capacity computation for the BEC is even simpler:

$$c_{BEC} = \max_{\mathbf{p}} \{= (1 - p_{CE})\eta\left(p\right)\} = 1 - p_{CE} \tag{4.33}$$

which is again atained when $p = 1 - p = 1/2$ (the BEC also is a *symmetric* channel). The Shannon capacity in this case is coincident with the *throughput* of the packet network when p_{CE} is interpreted as the probability of packet loss (drop).

We can refine a bit the modeling of packet communications by (re)introducing error events into the BEC model, that is, assuming that the capability of the CRC of detecting errors in a packet is limited, and that therefore with probability p_e some packets containing some errors will also be forwarded/used - we will call this channel BSC/E, whose model is shown in Fig. 4.7. Since the channel is symmetric, we assume from the start that the capacity is attained with equally probable source symbols ($H(S) = 1$) as shown in the figure; under such conditions, and considering symmetry again, the output probabilities are easily found:

$$P(\hat{0}) = P(\hat{1}) = \frac{1}{2}(1 - p_{CE}) \;\; , \;\; P(E) = p_{CE}$$

and we can find

$$c_{BSC/E} = H(S) - H(S|C) = 1 - H(S|\hat{0})P(\hat{0}) - H(S|E)P(E) - H(S|\hat{1})P(\hat{1})$$

$$= 1 - 2H(S|\hat{0})P(\hat{0}) - H(S|E)P(E) = 1 - (1 - p_{CE})H(S|\hat{0}) - p_{CE}H(S|E) \tag{4.34}$$

On the other hand,

$$P(0|\hat{0}) = \frac{P(\hat{0}|0)P(0)}{P(\hat{0})} = 1 - \frac{p_e}{1 - p_{CE}} \;\; , \;\; P(0|E) = \frac{P(E|0)P(0)}{P(E)} = 1/2$$

so that we have

$$c_{BSC/E} = 1-(1-p_{CE})\eta\left(\frac{p_e}{1 - p_{CE}}\right) - p_{CE} = (1-p_{CE})\left[1 - \eta\left(\frac{p_e}{1 - p_{CE}}\right)\right] \tag{4.35}$$

and the result is understandable: the capacity loss factor equal to $1 - p_{CE}$ resulting from erasures remains, this time applied to the capacity of the BSC with errors on the remaining fraction of unerased packets, i.e., $p_e/(1 - p_{CE})$.

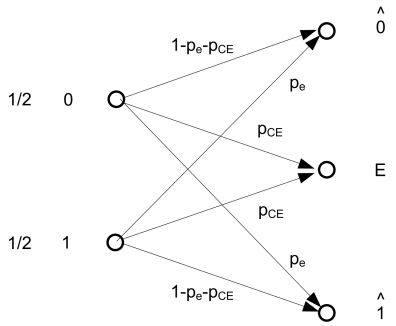


Figure 4.7 Binary symmetric channel with erasure BSC/E

We turn now to the Z channel. Its mutual information is

$$I(S,C) = \eta\left(p + p_z(1-p)\right) - (1-p)\eta(p_z) \tag{4.36}$$

and to find its maximum wrt p:

$$\frac{dI(S,C)]}{dp} = \eta'\left(p + p_z(1-p)\right)(1-p_z) + \eta(p_z)$$

From (3.19) and equating to 0 we get,

$$\log\frac{(1-p)(1-p_z)}{p + p_z(1-p)} = -\frac{\eta(p_z)}{1-p_z} \tag{4.37}$$

so that, after a few computations, we get

$$p = p_{cap} = \frac{1 - p_z(1+2^{-\beta})}{(1-p_z)\cdot(1+2^{-\beta})} \quad , \quad \beta \triangleq \frac{\eta(p_z)}{1-p_z} \tag{4.38}$$

The value p_{cap} leads of a capacity

$$c_Z = \eta\left(p_{cap} + p_z(1-p_{cap})\right) - (1-p_{cap})\eta(p_z)$$

$$= \eta\left(\frac{1}{1+2^{-\beta}}\right) - \frac{\beta}{1+2^{\beta}} = \eta\left(\frac{1}{1+2^{-\frac{\eta(p_z)}{1-p_z}}}\right) - \frac{\frac{\eta(p_z)}{1-p_z}}{1+2^{\frac{\eta(p_z)}{1-p_z}}} \tag{4.39}$$

When p_z is small, as it usually is, then $2^{\pm\beta} \simeq 1$, $\beta \simeq \eta(p_z)$ and the capacity is approximately

$$c_Z \simeq 1 - \frac{1}{2}\eta(p_z) \tag{4.40}$$

which formally resembles that of the BSC. The two capacities are compared in Fig. 4.8, and it is noted that the Z channel, not encompassing errors on 0, is better than the BSC.

Example 4.18

If the BSC is implemented thorugh a BPSK modem on AWGN with matched-filter hard-detection, then $p_e = Q(\sqrt{2E_s/N_0})$. In this case, we can then directly relate the signal-to-noise ratio E_s/N_0 to the usable capacity of the link (i.e., considering the channel as a BSC):

$$c_{BPSK} = 1 - \eta(p_e) = 1 - \eta\left[Q\left(\sqrt{\frac{2E_s}{N_0}}\right)\right] \tag{4.41}$$

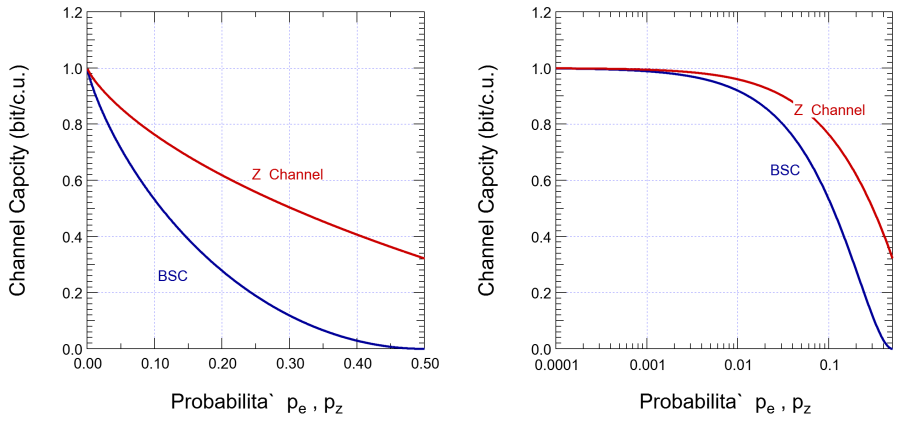


Figure 4.8 Comparison between the capacity of the BSC and the Z channel

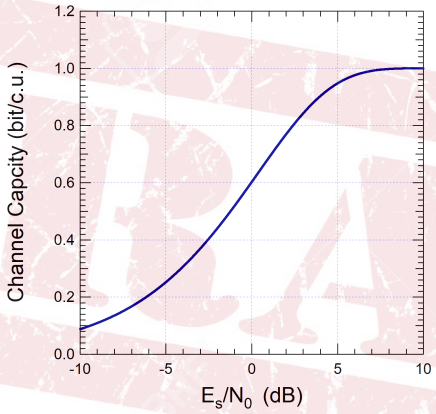


Figure 4.9 Capacity of the BSC implemented as a BPSK modem with AWGN and matched-filter hard-detection

We see in Fig. 4.9 that a non-negligible capacity can be achieved even for unexpectedly low signal-to-noise ratios, as low as -10 dB. We will come back to this when we discuss channel capacity with Gaussian noise and the Shannon efficiency plane.

Example 4.19

A binary interference channel is defined by

$$C[n] = S[n] + W[n] \tag{4.42}$$

where the source S is binary with levels ± 1, and the noise $W[n]$ is independent form S and is also binary, assuming the two levels $\pm\alpha$ with equiprobability. We can interpret $W[n]$ as an interference term akin to the useful signal, but with bearing a different amplitude.

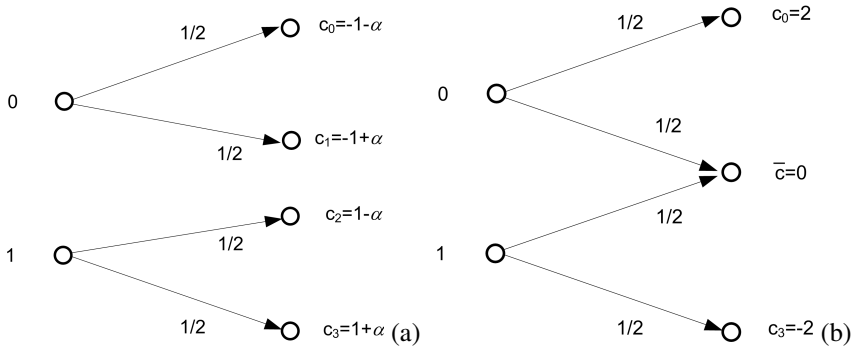


Figure 4.10 Binary interference channel

The channel alphabet has the 4 levels $\{c_0 = -1-\alpha, c_1 = -1+\alpha, c_2 = +1-\alpha, c_3 = +1+\alpha\}$, and the relevant channel probabilities are shown in Fig. 4.10 (a), where we also see that the channel is symmetric. The capacity will therefore be attained with $p = 1-p = 1/2$. Before computing the mutual information, we rmark that the four output symbols frfom the channel all occur with probability 1/4 since the source is equiprobable, and so are also the (source-independent) noise levels. Before explicitly computing the mutual information, let us evaluate the equivocation:

$$H(S|C) = \frac{1}{4}\sum_{k=0}^{3} H(S|c_k)p(c_k) = \frac{1}{4}\sum_{k=0}^{3} \eta[p(0|c_k)]p(c_k) \tag{4.43}$$

If $\alpha \neq \pm 1$ is known to the receiver, the uncertainties $\eta[p(0|c_k)]$ are all *zero*, and the channel capacity is *unity*: the channel is ideal. Despite interference, we can always perform correct detection of the transmitted symbol.

If we have $\alpha = 1$ instead, that is, if the interference is structurally *identical* to the source, the channel becomes as in Fig. 4.10 (b) with the symbols c_2 and c_3 collapsing onto a single symbol $\bar{c} = 0$ having $p(\bar{c}) = 1/2$, and things change: we obviously still have $H(S|c_0) = H(S|c_3) = 0$, but

$$p(0|\bar{c}) = \frac{p(\bar{c}|0)p(0)}{p(\bar{c}} = \frac{1/2 \cdot 1/2}{1/2} = \frac{1}{2} \tag{4.44}$$

and (recall that $H(S) = 1$ since $p = 1/2$)

$$c = 1 - H(S|C) = 1 - \eta\left(p(0|\bar{c})\right)p(\bar{c}) = 1 - \frac{1}{2}\eta\left(\frac{1}{2}\right) = \frac{1}{2} \text{ bit/c.u.} \tag{4.45}$$

The capacity is now reduced to 1/2 bit/c.u. because the channel is ideal with probability 1/2 (when the interference is constructive) and useless with probability 1/2 (when the interference is destructive, i.e., opposite to the source bit). The same happens when $\alpha = -1$. The reader can discuss the analogy of this example with the BEC case.

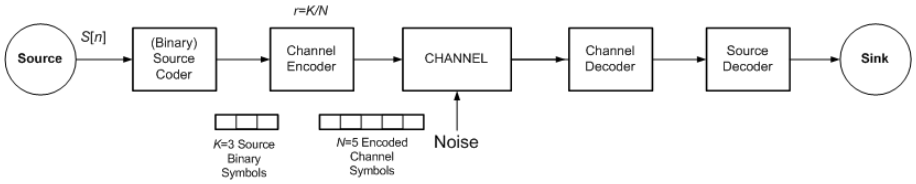


Figure 4.11 Digital link with channel coding

4.4 Shannon's channel coding theorem

4.4.1 Basics of channel coding

In the previous section, we defined the quantity c as the maximum value of $I(S, C)$, and we called this quantity "capacity"" suggesting an interpretation of the quantity as a kind of a limit to the use of the channel – without any apparent justification. In addition, calculation of the BSC capacity seems to involve a paradox: we found that c cannot be greater than 1 bit, while the source that attains capacity has a uniform distribution, generating 1 bit/symbol of information, that is, more than capacity itself. Can the channel support this? We must provide a clear answer to these two questions/doubts, specifying the actual meaning of c.

Take the BSC as an example for our discussion. On one side, we have the source, which generates information at the rate of $H(S)$ bits/symbol, and on the other side, we have the channel, with a capacity of c bits/symbol (or, in the parlance of information theory, bits/channel use or bits/c.u.). What happens if our source is such that $H(S) \geq c$ and we wish to transmit this information over the channel? Are we violating some fundamental law? From another, more practical standpoint, still considering the BSC, we are led to say: the error probability p_e of the channel certainly does not depend on the source, and remains *unchanged* whether $H(S) \geq c$ or vice versa, so what ? Is it really possible to improve the reliability of the noisy channel?

The answer to all of the above questions lies in the notion of *channel coding*, tha comes before transmission over the noisy channel, as shown in Fig. **??**. This system architecture is typical of the vast majority of the digital links of the modern Internet. The different criteria or algorithms for channel coding are very many, depending on the application – we will examine some of them in Chapter 5. In general, the source (information) bits to be sent over the noisy channel are "protected" from noise by adding additional redundancy symbols derived from the source bits themselves. The purpose of these symbols is to allow the decoder observing the noisy channel to compensate for possible channel errors generated by noise. Consider the example in Fig. **??**, in which the source bits are grouped into *blocks* of length K, and the encoder produces, for any such block, another other code blocks of binary digits with length $N > K$. Redundancy is

$$\eta = (N - K)/N = 1 - K/N \tag{4.46}$$

and the *coding rate* is

$$r \stackrel{\triangle}{=} 1 - \eta = K/N < 1 \tag{4.47}$$

As we will see in a whilem (and in much greater detail in Chapter 5), the presence of this redundancy allows the decoder to *detect* and *correct* any errors in the code block introduced by the BSC -— provided, of course, that the number of such errors is less than a certain limit determined by the type of the encoding and the length of the blocks.

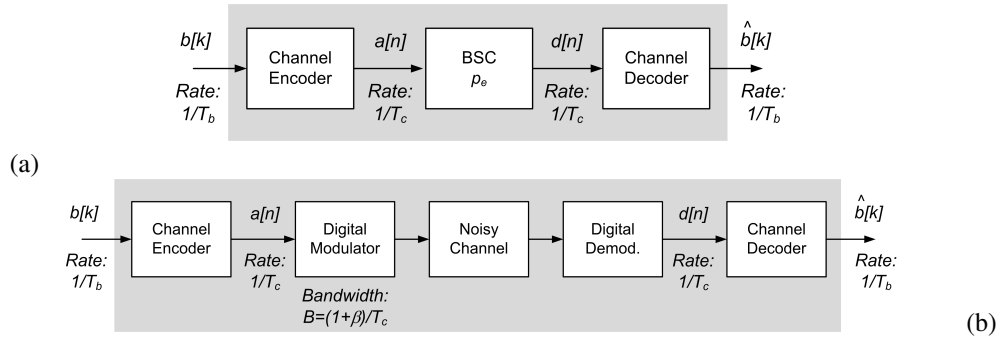


Figure 4.12 Improving the BER of a digital link via channel coding

4.4.2 Channel coding for dummies

We wish to improve the quality of a digital link on a BSC channel with a probability of error (Bit Error rate) p_e. The simplest approach is using a *repetition code*: to send out a bit of information $b[k]$, we send it twice, three times, ..., N times to make sure that the receiver better "understands" what is being transmitted. When applying this kind of encoding, we must understand the subtle difference between *bits* and *binary symbols*: using a code with $N-1$ repetitions, the n-th *encoded binary symbol* $a[n]$ is

$$a[n] = b[n//N] \tag{4.48}$$

where the // operator denotes the result of the integer division of n by N, and where $b[n//N] = b[k]$ represents the source (or information) bit. In (4.48), n is the index of the encoded symbol that runs at the (coded) symbol rate $R_c = 1/T_c$ (the subscript $_c$ stands for *coded*), while $k = n//N$ is the bit index that runs at the (slower) bit rate $R_b = 1/T_b$, as shown in Fig. 4.12 (a). It is easily understood that if we are forced to keep a constant signaling rate of $a[n]$ on the channel, the effect of the repetition code is to *decrease* the net information bit rate by a factor of N: $R_b = R_c/N$, since the $N-1$ binary symbols that we add specifically to the source bit do not provide any additional actual information - they represent our redundancy, since their only purpose is to imrpvoe the robustness of the link.

The redundancy introduced by channel coding is quantified by the coding rate.

$$r \stackrel{\triangle}{=} R_b/R_c < 1 \tag{4.49}$$

and/or the redundancy factor

$$\eta \stackrel{\triangle}{=} \frac{R_c - R_b}{R_c} = 1 - r \tag{4.50}$$

In our case, we added $N-1$ redundant symbols to each information bit, so $r = 1/N$ and $\eta = 1 - 1/N$. The smaller the rate, the closer the redundancy is to 1, and the relative information content of each encoded symbol becomes vanishingly smaller.

On the other hand, decreasing the coding rate has intuitively the effect of increasing the reliability of the received data. Let's quantify this improvement by calculating the *end-to-end BER* P_e, i.e., the probability of detecting a wrong information bit $b[k]$ after encoding and decoding. To evaluate this parameter, we are now to specify the algorithm

applied by the *decoder* that observes the (binary) symbols $d[n]$ at the output of the BSC in Fig. 4.12, to derive an estimate $\hat{b}[k]$ of the k-th information bit.

It can be shown that the optimum decoder for the repetition code applies *majority decoding*: starting from a block of N received (noisy) symbols $d[kN], d[kN+1], ..., d[kN+N-1]$ that encode the bit $b[k]$, the decoder returns $\hat{b}[k] = 0$ if the number of 0 symbols in the received block is greater than the number of 1s, vice versa $\hat{b}[k] = 1$. In a repetition code, N is usually odd, leading to no decoding ambiguities.

The rationale of majority decoding is simple. If all of the symbols received in code block are identical, the decoder has no alternative and decides for the value that has appeared in all of the repetitions. We understand that either there have been no errors at all on the channel, or there have been exactly N errors (i.e., all the received symbols are incorrect). The decoder obviously cannot tell these two cases apart, but we see that the second case, which produces a decoding error, is very low-probability, so that the decoder is almost always right. In all of the other cases where we have *fewer* than N errors and more than 0 errors, the decoder sees that there is no agreement between the received symbols and that therefore something has gone wrong: the decoder "detects up to $N-1$ channel errors". Unfortunately, if the majority of the received symbols are in error, the final decision on $b[k]$ will also be incorrect – even if errors has been detected, the decodeer is not able to correct them. Channel errors can be corrected as long as they represent the *minority*, that is, up to a number of $(N-1)/2$. In this case, the decoder decision will be correct, and the channel errors will have been detected *and* corrected.

Evaluating the end-to-end error probability P_e after decoding is simple. Let N_E be the random variable that represents the number of wrong symbols in a received codeword $d[kN], d[kN+1], ..., d[kN+N-1]$, $0 \leq N_E \leq N$; then

$$P_e = Pr\{N_E > (N-1)/2\} \tag{4.51}$$

In our channel model, errors on consecutive symbols are produced by the BSC independently from each other, so that the probability law of N_E is *binomial* with probability p_e (the error probability of the unprotected BSC):

$$\Pr\{N_E = n\} = \binom{N}{n} p_e^n (1-p_e)^{N-n} \quad 0 \leq n \leq N \tag{4.52}$$

so the probability of error, that is, the probability of having more than $(N-1)/2$ errors, is equal to the sum of the probabilities of having a number of errors equal to $(N+1)/2, (N+1)/2+1, ..., N$ (incompatible events):

$$P_e = \sum_{n=(N+1)/2}^{N} \binom{N}{n} p_e^n (1-p_e)^{N-n} \tag{4.53}$$

If $p_e < 1/2$, we will see that $P_e < p_e$ so that the channel code has actually reduced the probability of error on the information bit. In particular, our BSC could be the high-level model of a digital link with binary signaling in AWGN with optimum matched-filte/hard-limiter detection (Fig. 4.12 (b)), for which $p_e = Q(\sqrt{2E_s/N_0})$. We show in figure 4.13 the end-to-end BER curves P_e as a function of the E_s/N_0 ratio for different values of the coded block lenght N.

We see that, given a certain E_s/N_0 and thus given a certain uncoded p_e of the BSC, the coded BER P_e can be arbitrarily small as long as the number N of repetitions is adequately

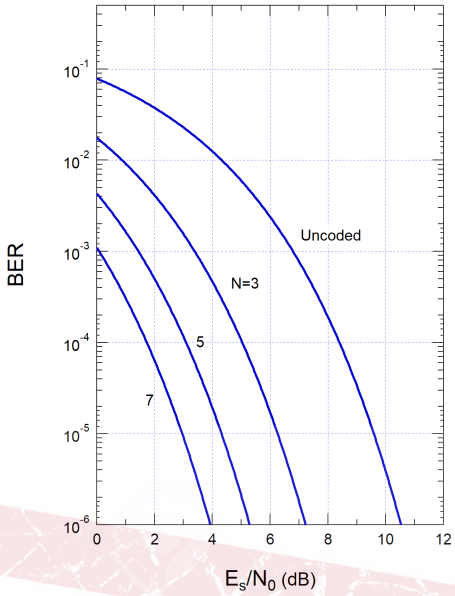


Figure 4.13 BER of repetition coding with majority decoding on hard-detected BPSK/AWGN channels

increased (i.e., the rate $r = 1/N$ is proportionally reduced) – a good property for an error-correcting code! What is the price to pay for this? Assume that communications on the physical channel is performed with a modem whose signaling rate on the channel R_c cannot be changed (it is a specification of the piece of HW). If we need to significantly increase N (decrease r) to improve the coded BER P_e, we obtain an increasingly smaller net information bit rate $R_b = r \cdot R_c$ – we have increased reliability at the expense of the achieved bit rate.

Example 4.20

We wish to derive the BER after repetition coding on a BSC as in Fig. **??** (b). As an example, we start with a relatively bad BSC having p_e=0.01 (one wrong information bit every 100 transmitted). The dominant term in the BER (4.53) is the first in the sum, that is, the one with the smallest power of p_e; we can also assume that $(1 - p_e) \simeq 1$ for each term since $p_e \ll 1$. According to this approximation,

$$P_e \simeq \binom{N}{\frac{N+1}{2}} p_e^{(N+1)/2} \tag{4.54}$$

Using Stirling approximation for the binomial coefficient we get

$$P_e \simeq \frac{2^N}{\sqrt{\pi(N+1)/2}}\, p_e^{(N+1)/2} = \sqrt{\frac{2p_e}{\pi(N+1)}}\, (2\sqrt{p_e})^N \tag{4.55}$$

We see that as soon as $2\sqrt{p_e} < 1$, the end-to-end BER after the code decreases exponentially according to $N/2$, so that we can attain an arbitrarily small BER values for an adequately large value of N.

The improvement of the link quality with channel coding is impressive. Let us assume that we need to keep net bit rate R_b constant after applying the code. It is then necessary to increase the signaling rate of the encoded symbols $R_c = R_b/r$, at the cost of proportionally increasing the bandwidth required on the physical channel to support the same bit rate R_b. For example, using a SRRC-type signaling pulse with roll-off factor β and binary alphabet, the (radio frequency) bandwidth is $B_{RF} = (1+\beta)R_c = (1+\beta)R_b/r$, wider than in the uncoded by a factor $1/r$. We will return to this topic in the 5.6 section.

4.4.3 Shannon's coding theorem

We now postpone the (more) detailed study of channel codes to Chapter 5, and we ask ourselves a fundamental question, related to the topic of Shannon capacity: is it always true that to (arbitrarily) reduce the BER we must (arbitrarily) reduce the coding rate and therefore the spectral efficiency as happens with the repetition code? Are there better codes? In European digital satellite television DVB-S2, the E_s/N_0 ratio at the receiver is unfavorable (just a few dB), but the application is very demanding from the BER point of view: good digital video requires BER$= 10^{-10}$, a condition known as *Quasi-Error-Free* (QEF) tro prevent image freezing and/or distortion. Is it really necessary to resort ot very low coding rates to guarantee an acceptable BER?

To understand this, consider again the transmission of a memoryless source on our BSC with capacity $c = 1 - \eta(p_e)$, and assume that we are using a channel code obeying the condition $r \leq c$. In this case, the encoded binary symbols each carry an amount of information equal to $rH(S)$, so that, even assuming that the entropy of the source is maximum, $H(S) = H_{MAX} = 1$, the information associated with each of the symbols input to the channel will always be $\leq c$ since we have chosen an appropriate rate r. We will see in a while that such condition is essential to attain *reliable communication*.

How can we construct a code with an assigned code rate r? We extend the notion of a repetition code by introducing a general *block code*, which maps finite sequences (packets) of K symbols of source bits **b**, called blocks, into finite binary sequences (blocks) of lenght N of channel (encoded) binary symbols, $N > K$ – we will consider both **b** and **a** as *row* vectors of binary symbols. Defining the code means mapping the 2^K possible source blocks **b** of length K into a *subset* of cardinality 2^K (the so-called *codebook* Γ) of the complete set Ω_2^N of the 2^N binary words (channel words) of length N, with $K/N = r < 1$, r encoding rate, as shown in Fig. 4.14. Choosing just "a few" 2^K codewords **a** within the "huge" space with the "many" 2^N channel words allows the codewords to be very sparse, so that they can be selected very "far apart", i.e., very different, from each other. When a codeword is sent over the noisy channel, some of its symbols may be flipped, turning into into a different word (one of the 2^N channel words of Ω_2^N) – in spite of this, the transmittede codeword may still be recognizable, because with just a few channel errors the transmitted codeword has "moved" a bit in Ω_2^N (has changed a bit) but not so much, the other codewords they are sufficiently far apart (different), and so it can still be recognized by the decoder. The repetition code is a block code with $K = 1$ and $r = K/N = 1/N$. By choosing K, N, and the codebook appropriately, we can create robust (good) codes at the desired r rate, as we will see in Chapter 5.

A peculiar strategy for defining a code as in Fig. 4.14, is the one envisioned by Shannon in the proof of the so-called *channel coding theorem*: instead of worrying about spacing the words in the codebook as far apart as possible, we instead choose these words *randomly*,

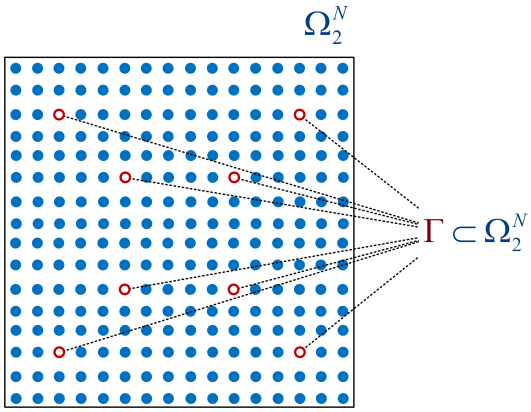


Figure 4.14 Construction of a block code

more specifically following the probability distribution of the channel input symbols that attains the capacity of the channel used. This is highly counterintuitive, contradicting the approach of placing codewords as spaced apart as possible. The *random code* thus obtained may be good or bad with a certain probability, meaning by "good" the fact that it actually guarantees an error probability $P(E_m)$ on the source bits smaller than a certain value ε set by the application (audio, voice, video) for which we are using the coded link. More specifically, different random codes feature different BER $P(E_m)$ after coding. Considering them all, we understand that the BER of a random code is... a random variable, very small for good codes, larger for bad codes.

The reason for resorting to random codes is the following: it can be shown that the expected value $E\{P(E_m)\}$ of the error probability of all these random codes may be "good," that is, it is less than the value ε that we need, regardless of how small ε may be. This happens as long as K (and therefore also N) is sufficiently large, without the need to make $r = K/N$ vanishingly small: the only condition to be met for this to happen is what we may imagine, i.e, with $K/N = r < c$ fixed, and . In the reasoning and the proof above, we considered the *expected value* of $P(E_m)$. If such *average* $E\{P(E_m)\}$ over all codes of $P(E_m)$ can be reduced smaller than any ε, then there must exist *at least one* code that individually enjoys the same property, that is, that $P(E_m) < \varepsilon$!

We now know that, for any *finite* $r \leq c$, there certainly exists an error-protecting code with that rate that allows the BER on the source bits to be reduced to an arbitrarily small value that guarantees *reliable communication* – Shannon's coding theorem (whose detailed proof is too complicated to be reported here). Unfortunately, it is an *existence* theorem, not suggesting how to actually find a good enough code for a certain application. The ICT engineer must try to get as close as possible to this performance limit with practically implementable channel coding schemes: the subject of the next chapter.

There is also a converse version of the theorem: if $r > c$, then there is no code whose $P(E_m)$ can be reduced to an arbitrarily small value – there will always be a *floor* to the $P(E_m)$ that will be irreducible, regardless of the technology. The converse channel coding theorem also says that if one is satisfied with a given finite target BER $P(E_m) = p_T$, then it is possible to increase the rate of the code up to the value

$$r = \frac{c}{1 - \eta(p_T)} \tag{4.56}$$

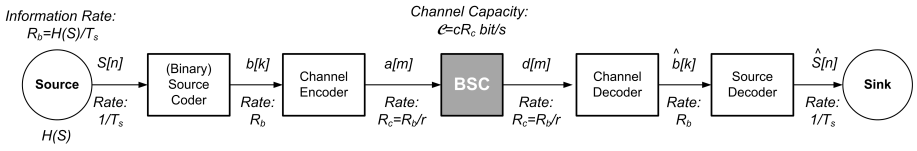


Figure 4.15 Information rate and capacity $\mathcal{C}$

which turns out to be greater than the "reliable" capacity c.

The real-time information rate R_b of a source with entropy $H(s)$ bits/symbol that generates symbols at a rate $R_s = 1/T_s$ symbols/s is $R_b = H(S)/T_s$. Considering Fig. 4.15, we can see that the reliability condition on capacity can also be expressed as

$$R_b \leq \mathcal{C} \tag{4.57}$$

where $\mathcal{C} = c \cdot R_c$ is real-time Shannon capacity in bit/s when the communication channel is used at a rate R_c coded symbols/s or c.u./s.

We see now the relevance of the notion of mutual information to derive Shannon capacity and, through the channel coding theorem, the consequent condition of reliable communication.

Example 4.21

The converse channel coding theorem allows us to establish the *performance limit* that a given technology can provide, assuming that a certain residual BER may remain after channel decoding, depending on the application. For example, digitized voice is still acceptable with BER=10^{-3}, while HD video requires BER= 10^{-10}, so the reliability requirements can be very different and, in any case, *finite*.

Consider a BSC channel implemented through a BPSK modem, with capacity given by (4.41). Taking into account that $E_s = rE_b$, from the inverse theorem (4.56) and from (4.41), we have

$$r = \frac{1 - \eta\left[Q\left(\sqrt{\frac{2rE_b}{N_0}}\right)\right]}{1 - \eta(BER)} \tag{4.58}$$

so that we can derive a limit-curve for the BER as follows:

$$BER = \eta^{-1}\left(1 - \frac{1 - \eta\left[Q\left(\sqrt{\frac{2rE_b}{N_0}}\right)\right]}{r}\right) \tag{4.59}$$

The result (derived numerically since the function η^{-1} is not closed-form) is shown in Fig. 4.16, in the form of several BER limit curves for different coding rates r. The value of E_b/N_0 for which the curves become "vertical" corresponds to the case of arbitrarily small BER, that is, it provides the value for E_b/N_0 for which the capacity c is exactly equal to the value of r of the curve.

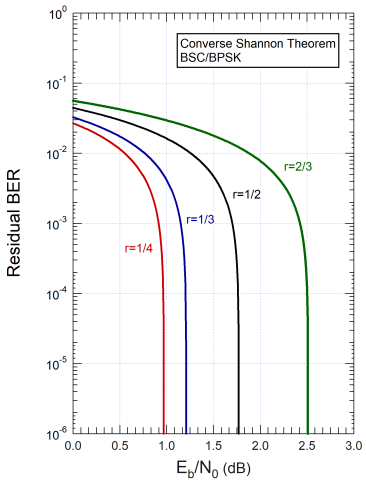


Figure 4.16 Limit curve for the BSC/BPSK channel

4.5 The AWGN channel

4.5.1 "Soft" and "hard" channels

Up to now, we have considered a very abstract model of a discrete transmission channel, that is, one with a finite input and output alphabet. This is for instance the case of the systems outlined in Figs. 4.2 (a) and (b), in which "regeneration" (hard-limiting) of the received signal has taken place. However, we can consider the "decision variable" $z[n]$ before hard-limiting takes place as in Fig. 4.2 (c) and therefore we can model the channel as having a binary input (the k-th source bit) and a *continuous output* (in the signal-theoretic sense, i.e., continuous amplitude). A continuous-amplitude output is sometimes referred to as "soft" output, as opposed to the "hard" output of the hard limiter. It is intuitive that the channel with a "soft" output may prove more promising from the point of view of channel capacity. In fact, reducing a quantity from infinitely many levels, as in the case of a soft output, to only two levels, as in the case of a hard output, seems to result in some loss of information. This is fully confirmed by the practice if we add a channel encoder to our transmission scheme. As we will see in the section 5.2, given the same channel code, we can decode the source message either at the output of a soft or of a hard channel, with decoding algorithms of comparable complexity. Soft-amplitude decoding shows a robustness to noise about 2 dB greater than threshold decoding, meaning that the end-to-end error probability P_e with soft-input decoding with a given noise level is equal to that obtained with hard-input decoding with a noise level 2 dB lower.

This observation also leads us to reconsider the transmitter side of the channel: if we use a channel model closer to the physical layer, we recognize that what is actually sent to transmit the source information is a *waveform*, something that is continuous in amplitude and in time, obtained with some coding and modulation method. Considering waveforms rather than symbols is particularly significant when the link bears some limitations in terms of the available bandwidth for communication, an aspect that is not considered at all in the higher-layer discrete-channel model. To approach the issue of communications over a

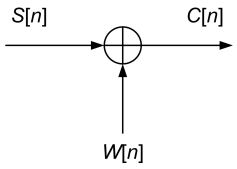


Figure 4.17 Discrete-time Additive White Gaussian Noise (AWGN) channel

bandlimited channel, we must proceed step by step, starting from the simpler consideration of a discrete-time channel with continuous (soft) input/output amplitudes.

A representative channel model in many cases is the one outlined in Fig. 4.17, in which transmission of the soft source symbol $S[n]$ is disturbed by the presence of an additive white Gaussian noise (AWGN) component $W[n]$, so that the channel output is $C[n] = S[n] + W[n]$. To properly considering a soft (continuous) source, we need to rethink the definition of entropy and mutual information. information to be associated with that quantity. Our continuous, stationary, memoryless source $S[n]$ is now be characterized by a *probability density function* (pdf) $f_S(s)$. Trying to extend the previous definition of entropy, we "model" this continuous source as a discrete source $S_\Delta[n]$ with infinitely many equally-spaced quantized values $k\Delta s$ with a certain quantization step Δs. In this cASE, The entropy is

$$H(S_\Delta) = \sum_{m=-\infty}^{\infty} (f_S(m\Delta s)\Delta s) \log\left(\frac{1}{f_S(m\Delta s)\Delta s}\right)$$

$$= \sum_{m=-\infty}^{\infty} f_S(m\Delta s) \log\left(\frac{1}{f_S(m\Delta s)}\right) \cdot \Delta s + \log\left(\frac{1}{\Delta s}\right) \sum_{m=-\infty}^{\infty} f_S(m\Delta s) \cdot \Delta s \quad (4.60)$$

We now let $\Delta s \to 0$ to get back to the soft source, and we see that the two summations tend to finite results. In particular, the second summation tends to 1 irrespective of any particular $f_S(s)$. Unfortunately the quantity $\log(1/\Delta s)$ tends to $+\infty$, and the purported extended definition of entropy is useless, for a very good reason. A source with a finite number of levels is associated with finite average information that can be represented with a finite number of bits equal, by definition, to the entropy $H(S)$. A continuous source with "infinitely many" levels is associated with *infinite* information that can only be represented with an infinite number of digits, as our definition correctly suggests. How to get out of this impasse?

When computing the mutual information between source and channel, we are dealing with a *difference* between entropies (or similar quantities), specifically between the source entropy and the equivocation: $I(S,C) = H(S) - H(S|C) = H(C) - H(C|S)$. Imagine sending a certain soft value s_0 of $S[n]$, that is, a real number with infinite digits (decimal or binary depending on the representation) and therefore carrying, in theory, an infinite number of bits of information. The output of the channel $C[n]$, being soft, will also carry infinite entropy $H(C)$ in its turn. In our case of an AWGN channel (but this reasoning can be generalized) and *conditionally* on $S[n] = s_0$, we have that $C[n] = s_0 + W[n]$ therefore (considering that s_0 is a known value) the conditional entropy $H(C|S = s_0)$ ì essentially equal to $H(W)$ and is therefore still *infinite*. Consequently, the mutual information is an indeterminate form $\infty - \infty$ that can be any value, including a *finite* one.

To sort this out, we also quantize the output of the channel $C[n]$ with a step Δc, and we write the expression for the mutual information (4.9) by adapting the definitions of the

n	$C[n]$ =2.3503858....+$W[n]$
0	2.3502098...
1	2.3502087...
2	2.3500982...
3	2.3502089...
4	2.3504389...
5	2.3501468...
6	2.3500938...

Table 4.1 Heuristic representation of $I(S,C)$

probabilities as already done in (4.60):

$$I(S,C) \simeq \sum_k \sum_m f_{SC}(m\Delta s, k\Delta c)\Delta s\Delta c \cdot \log\left(\frac{f_{SC}(m\Delta s, k\Delta c)\Delta s\Delta c}{f_S(m\Delta s)\Delta s \cdot f_C(k\Delta c)\Delta c}\right)$$

where $f_{SC}(s,c)$ è is the joint pdf of $S[n]$ and $C[n]$. Developing we get

$$= \sum_k \sum_m f_{SC}(m\Delta s, k\Delta c) \log\left(\frac{f_{SC}(m\Delta s, k\Delta c)}{f_S(m\Delta s) \cdot f_C(k\Delta c)}\right) \Delta s\Delta c$$

$$\xrightarrow{\Delta c, \Delta s \to 0} \int\int f_{SC}(s,c) \log\left(\frac{f_{SC}(s,c)}{f_S(s) \cdot f_C(c)}\right) dsdc \tag{4.61}$$

and we find a nice surprise: the result is perfectly well-defined and finite, and constitutes the new definition of mutual information $I(S,C)$ for continuous (soft) input and output channels.

$$I(S,C) \stackrel{\triangle}{=} \int\int f_{SC}(s,c) \log\left(\frac{f_{SC}(s,c)}{f_S(s) \cdot f_C(c)}\right) dsdc \tag{4.62}$$

How should we interpret this result, namely, that the mutual information of the soft-input soft-output channel is finite?

Example 4.22

Imagine setting a certain source value s_0, for example, $s_0 = 2.3503858....$, and repeating the transmission of $S[n] = s_0$ on the AWGN channel many times, in a case where $\sigma_w \ll s_0$; we obtain a series of values like those in Table 4.1, in which we note that the first digits of $C[n]$ (the most significant ones) are "stable", while the subsequent digits (the least significant ones) change randomly due to noise (why don't the most significant ones vary?). The table is essentially the representation of the quantities we have just discussed: all the values of $C[n]$ allow us to evaluate its uncertainty, that is, the entropy $H(C)$, which is infinite because it affects the infinite least significant decimal digits; the finite number of "stable" (certain) digits, represent the finite information about the value of the source obtained by observing $C[n]$, that is, the $I(S,C)$; finally, the least significant digits, completely uncertain, represent the $H(C|S)$, that is, the residual uncertainty about the output after removing the input information (in particular, the stable digits).

If we now apply the same steps as in (4.9) to the definition (4.62), but in reverse, we easily obtain the following relation:

$$I(S,C) = \int f_S(s) \log\left(\frac{1}{f_S(s)}\right) ds - \int\int f_{SC}(s,c) \log\left(\frac{1}{f_{S|C}(s)}\right) dsdc$$

$$\stackrel{\Delta}{=} h(S) - h(S|C) \tag{4.63}$$

where we defined two new quantities, formally similar to $H(S)$ and $H(S|C)$, which we call *differential entropy* and *differential equivocation*, respectively. We also define $h(C)$ and $h(C|S)$ the same way.

Why do we call *differential* the quantities we've just defined ? In general, if we want to evaluate the *difference* between the entropy $H(S)$ and that $H(T)$ of a second source $T[n]$, starting from the entropies of the two quantized sources $S_\Delta[n]$ and $T_\Delta[n]$ (and using the same step Δs for both), we obtain:

$$H(S_\Delta) - H(T_\Delta) =$$

$$\sum_{m=-\infty}^{\infty} f_S(m\Delta s) \log\left(\frac{1}{f_S(m\Delta s)\Delta s}\right) \cdot \Delta s + \log\left(\frac{1}{\Delta s}\right) \sum_{m=-\infty}^{\infty} f_S(m\Delta s) \cdot \Delta s$$

$$= \sum_{m=-\infty}^{\infty} f_T(m\Delta s) \log\left(\frac{1}{f_T(m\Delta s)\Delta s}\right) \cdot \Delta s + \log\left(\frac{1}{\Delta s}\right) \sum_{m=-\infty}^{\infty} f_T(m\Delta s) \cdot \Delta s \tag{4.64}$$

When $\Delta s \to 0$, the two terms tending to ∞ cancel each other, and the final result is just the (finite) difference

$$\lim_{\Delta s \to 0} \left\{ \sum_{m=-\infty}^{\infty} f_S(m\Delta s) \log\left(\frac{1}{f_S(m\Delta s)}\right) \Delta s \right\}$$

$$- \lim_{\Delta s \to 0} \left\{ \sum_{m=-\infty}^{\infty} f_T(m\Delta s) \log\left(\frac{1}{f_T(m\Delta s)}\right) \Delta s \right\}$$

$$= \int_{-\infty}^{\infty} f_S(s) \log\left(\frac{1}{f_S(s)}\right) ds - \int_{-\infty}^{\infty} f_T(s) \log\left(\frac{1}{f_T(s)}\right) ds \tag{4.65}$$

that can be computed through differential entropies. The unlimited fraction of entropy is common to the two sources for the mere fact that they are *soft* and is not different between the two - the individual part that characterizes them, and which remains in the difference, is just the differential entropy.

To sum up, whenever we wish to derive mutual information, and consequently Shannon capacity, involving soft quantities, we must use the *differential entropy*

$$h(S) \stackrel{\Delta}{=} \int_{-\infty}^{\infty} f_S(s) \log\left(\frac{1}{f_S(s)}\right) ds \tag{4.66}$$

keeping in mind that, unlike the case of a discrete source, it *does not* represent the information generated by the source, as we will see later on.

Example 4.23

Let us compute the differential entropy of a source $S[n]$ uniformly distributed in the (finite) interval $[s_1; s_2]$. By definition,

$$h(S) = \int_{s_1}^{s_2} f_S(s) \log\left(\frac{1}{f_S(s)}\right) ds = \int_{s_1}^{s_2} \frac{1}{s_2 - s_1} \log(s_2 - s_1)\, ds = \log(s_2 - s_1) \tag{4.67}$$

If $s_2 - s_1 < 1$, we note that $h(S) < 0$, therefore, as anticipated, we conclude that differential entropy *cannot* represent the information content of the source.

Let us now formulate a second problem: among all the probability densities with finite support, that is, different from zero only in the interval $[s_1; s_2]$, which is the one with maximum $h(S)$? We must find the maximum of

$$\int_{s_1}^{s_2} f_S(s) \log\left(\frac{1}{f_S(s)}\right) ds$$

under the normalization constraint $\int f_S(s)\, ds = 1$ (otherwise the question makes no sense).The solution is found through the calculus of variations. We form the Lagrangian function

$$\begin{aligned} \mathcal{L}(f_S, \lambda) &= \int_{s_1}^{s_2} f_S(s) \log\left(\frac{1}{f_S}(s)\right) ds + \lambda \left(\int_{s_1}^{s_2} f_S(s)\, ds - 1\right) \\ &= \int_{s_1}^{s_2} \left(f_S(s) \log\left(\frac{1}{f_S(s)}\right) + \lambda f_S(s)\right) ds - \lambda \end{aligned} \tag{4.68}$$

and, to find its maximum, we formally differentiate with respect to f_S, understood as an unknown variable, and equate it to 0:

$$\int_{s_1}^{s_2} \left(-1 + \log\left(\frac{1}{f_S(s)}\right) + \lambda\right) ds = 0$$

Given the arbitrariness of λ, we must have

$$-1 + \log\left(\frac{1}{f_S(s)}\right) + \lambda = 0 \quad \Rightarrow \quad f_S(s) = 2^{\lambda - 1}$$

that is, the probability density must be *constant* across the support interval. Using the normalization constraint $\int f_S(s)\, ds = 1$, we obtain the uniform probability density that we have already studied.

The result of the 23 example is not surprising: the uniform density is the one that maximizes the uncertainty about the source value, and in a certain sense it is the continuous equivalent of the equally probable levels of a discrete source.

Example 4.24

Let's now calculate the differential entropy for a Gaussian source with mean η_S and variance σ_S^2, for which

$$f_s(s) = \frac{1}{\sqrt{2\pi\sigma_S^2}} \exp\left(-\frac{(s-\eta_S)^2}{2\sigma_S^2}\right)$$

From the definition,

$$h(S) = \int_{-\infty}^{\infty} f_s(s) \cdot \left(\frac{1}{2}\log(2\pi\sigma^2 + \frac{(s-\eta_S)^2}{2\sigma_S^2}\log(e)\right) ds$$

$$= \frac{1}{2}\log(2\pi\sigma_S^2) + \frac{\sigma_S^2}{2\sigma^2}\log(e) = \frac{1}{2}[\log(2\pi\sigma_S^2) + \log(e)] = \log\left(\sqrt{2\pi e\sigma_S^2}\right) \quad (4.69)$$

We see that $h(S)$ is larger, the larger the variance of the source, and that (most importantly) it is *independent of the particular mean value* η_S.

In the previous example, we analyzed the case of finite support, that is, a constraint on the range of variation of the values taken by the source – in practice, a limitation on the peak value of the signal. In other cases, however, a constraint on the *power* of the source is more relevant. What, then, is the probability density that maximizes the differential entropy without any particular amplitude limitations on the signal, but with a power constraint of σ^2 (assuming a zero mean value)?

We proceed as in the previous example with variational calculus, now taking into account *two* constraints:

$$\int_{\infty}^{\infty} f_S(s)\log\left(\frac{1}{f_S(s)}\right) ds = \max \quad \text{con} \quad \int_{-\infty}^{\infty} s^2 f_S(s)\, ds = \sigma^2 \quad \text{e} \quad \int_{-\infty}^{\infty} f_S(s)\, ds = 1 \quad (4.70)$$

the Lagrangian is

$$\mathcal{L}(f_s, \lambda, \mu) = \int_{-\infty}^{\infty} \left[f_S(s)\log\left(\frac{1}{f_S(s)}\right) + \lambda s^2 f_S(s) + \mu f_S(s)\right] ds = \max \quad (4.71)$$

Differentianing f_S under the integral we get

$$\int_{-\infty}^{\infty} \left[-1 - \log(f_S(s)) + \lambda s^2 + \mu\right] ds = 0 \quad (4.72)$$

or, kepping into account the arbitrariness of λ and μ,

$$-1 - \log(f_S(s)) + \lambda s^2 + \mu = 0 \quad \Rightarrow \quad f_S(s) = \exp(-1+\mu)\exp(\lambda s^2) \quad (4.73)$$

that is, exponential of a quadratic. Applying the constraints, we find

$$f_S(s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{s^2}{2\sigma^2}\right) \quad (4.74)$$

i.e., the famililar Gaussian density.

The differential entropy $h(S)$ of a continuous source features some remarkable properties: for example, the joint entropy $h(S, Z)$ of two independent sources is equal to the sum

$h(S) + h(Z)$. Furthermore, as demonstrated in particular in Example 24, $h(S)$ is invariant under changes in the mean value of the source (the general proof is elementary: calculate $h(Z)$ with $Z[n] = a + S[n]$, a being a known constant).

4.5.2 The discrete-time AWGN channel

We now extend the concepts already introduced for the discrete channel to the continuous channel. A channel with continuous input and output is characterized as soon as the probability density $f_{C|S}(c|s)$ of the channel output is known, conditioned on a certain value of the channel input – something similar to the well-know channel probabilites. In the additive Gaussian channel in Fig. 4.17, we know that the noise has zero mean and variance σ_W^2, so we trivially have

$$f_{C|S}(c|s) = \frac{1}{\sqrt{2\pi\sigma_W^2}} e^{-\frac{(c-s)^2}{2\sigma_W^2}} \tag{4.75}$$

We already know that

$$I(S,C) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{SC}(s,c) \log\left(\frac{f_{SC}(s,c)}{f_S(s)\cdot f_C(c)}\right) dsdc$$

$$I(S,C) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_C(c|s) f_S(s) \log\left(\frac{f_C(c|s)}{\int_{-\infty}^{\infty} f_C(c|s) f_S(s)\, ds}\right) dsdc \tag{4.76}$$

The channel capacity is found by varying the source probability distribution $f_S(s)$ so that the mutual information is maximized:

$$c \triangleq \max_{f_S} I(S,C) = \max_{f_S} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_C(c|s) f_S(s) \log\left(\frac{f_C(c|s)}{\int_{-\infty}^{\infty} f_C(c|s) f_S(s)\, ds}\right) dcds \tag{4.77}$$

The channel coding theorem also applies to soft-input soft-output channels, just with this definition of capacity.

Finding a general solution to this equation is hopeless. For the AWGN channel, however, things are quite simple. For this case, we have $C[n] = S[n] + W[n]$, where we assume that the source $S[n]$ has arbitrary pdf $f_S(s)$ with zero mean and a known variance σ_S^2 (the zero-mean signal power). The noise $W[n]$ is Gaussian with zero mean and variance σ_W^2 (the noise power). The signal-to-noise ratio is $SNR = \sigma_S^2/\sigma_W^2$.

We know that

$$I(S,C) = h(S) - h(S|C) = I(C,S) = h(C) - h(C|S) \tag{4.78}$$

Computing $h(C|S)$ is simple: conditioned on a certain value s_0 of the source, the channel output is

$$C[n] = s_0 + W[n] \tag{4.79}$$

that is, s_0 is the *average value* of the output of channel $C[n]$. The output is (conditionally) Gaussian with a differential entropy that *does not* depend on its average value:

$$h(C|s_0) = \log\left(\sqrt{2\pi e\sigma_W^2}\right) \tag{4.80}$$

Averaging further over S and *whatever the (unknown) probability density* of $S[n]$ is, the result is

$$h(C|S) = \int \log\left(\sqrt{2\pi e\sigma_W^2}\right) f_S(s)ds = \log\left(\sqrt{2\pi e\sigma_W^2}\right) \tag{4.81}$$

That does not depend on the characteristics of the source (a general result for all additive channels). We can now write

$$c_{AWGN} \stackrel{\triangle}{=} \max_{f_S} I(S,C) = \max_{f_S}\left[h(C) - h(C|S)\right] = \max_{f_S} h(C) - \log\left(\sqrt{2\pi e\sigma_W^2}\right) \tag{4.82}$$

We must therefore seek for the maximum of $h(C)$, knowing that the variance of $C[n]$, given that $S[n]$ and $W[n]$ are independent, is $\sigma_C^2 = \sigma_S^2 + \sigma_W^2$. From Example 24 we know that the maximum of $h(C)$ among all probability densities a having variance σ_C^2 occurs if and when the output $C[n]$ is Gaussian, and it is $\log\sqrt{2\pi e\sigma_C^2}$. However, we must ask ourselves whether this condition is actually achievable - the answer is simple because, given that $W[n]$ is Gaussian and independent of $S[n]$, we obtain a Gaussian $C[n]$ when (also) the input $S[n]$ is Gaussian (which is therefore our *capacity-achieving distribution*), and the capacity is

$$c_{AWGN} = \log\left(\sqrt{2\pi e\left(\sigma_S^2 + \sigma_W^2\right)}\right) - \log\left(\sqrt{2\pi e\sigma_W^2}\right) =$$

$$= \frac{1}{2}\log\left(1 + \frac{\sigma_S^2}{\sigma_W^2}\right) \quad \text{(bit/c.u.)} \tag{4.83}$$

It is interesting to discuss the limiting cases: if the noise has a variance (power) that is very large compared to that of the source (signal), we note that the channel capacity tends to zero and we have the usual situation of the useless channel. Conversely, if the noise has a variance that is very small, the channel tends to become ideal, the SNR tends to grow unboundedly, and so does its logarithm. The conclusion is that the channel capacity grows indefinitely, unlike the case of the discrete channel, where it is limited by the size of the channel alphabet – a consequence of the input being soft. As already mentioned, it would theoretically be possible in the absence of noise to encode an arbitrarily long bit string into a *single* source symbol with an arbitrarily large number of decimal places. This value (symbol) could then be transmitted in a single channel use and recovered at the receiver and without any equivocation. In such a way, an arbitrarily large amount of information could be transmitted in a singe channel use: infinite channel capacity. The presence of noise places a limit to the accuracy in the estimation of the transmitted soft value: some of the less significant decimal places become uncertain, and the amount of information that can be transmitted becomes limited.

The capacity, that is, the quality of the channel, is directly linked to the signal-to-noise ratio (SNR):

$$c_{AWGN} = \frac{1}{2}\log\left(1 + \text{SNR}\right) \text{(bit/c.u.)} \tag{4.84}$$

The Gaussian channel can have a capacity of several bits/c.u., depending on the SNR at the receiver. This capacity must be exploited, as we will see in more detail below, through

appropriate Coding and Modulation (COD/MOD) methods.

Example 4.25

We introduce into the AWGN channel in Fig. 4.17 a random variability with Rayleigh statistics of the useful signal amplitude, independent of $W[n]$, according to the modeling of the wireless channel with flat fading that we will see in detail in the section 4.1.3):

$$C[n] = A \cdot S[n] + W[n] \tag{4.85}$$

where the probability density of the amplitude A is

$$f_A(a) = 2a \cdot \exp\left(-a^2\right) \quad , \quad a > 0 \tag{4.86}$$

with power $\mathrm{E}\{A^2\} = 1$. This model describes the case when the amplitude of the useful signal varies (very) slowly over time with respect to the symbol rate. In such a situation, we can regard Shannon capacity as a quantity that varies over time: in certain periods it will prove satisfactory, in others will not, depending on the value of the channel amplitude in that period of time. Given a certain value of the amplitude $A = \alpha$, the *conditional* capacity is

$$c(\alpha) = \frac{1}{2}\log\left(1 + \alpha^2 \frac{\sigma_S^2}{\sigma_W^2}\right) = \frac{1}{2}\log\left(1 + \alpha^2 \cdot \mathrm{SNR}\right) \tag{4.87}$$

Once we have defined a certain "minimum" desired capacity level c_0, we might be interested in calculating the fraction of time within which c is satisfactory, that is, in which $c \geq c_0$. By ergodically calculating the fraction of time as probability, we immediately find

$$\begin{aligned}\Pr\{c \geq c_0\} &= \Pr\left\{\frac{1}{2}\log\left(1 + A^2\mathrm{SNR}\right) \geq c_0\right\} \\ &= \Pr\left\{A \geq \sqrt{\frac{2^{2c_0} - 1}{\mathrm{SNR}}}\right\} = 1 - \frac{2^{2c_0} - 1}{\mathrm{SNR}}\end{aligned} \tag{4.88}$$

This quantity is the probability of the link being available (i.e., bearing a capacity greater than the minimum), and its 1's complement, i.e., $(2^{2c_0} - 1)/\mathrm{SNR}$, is the probability of being out of service (outage probability). The SNR value is the signal-to-noise ratio on the equivalent AWGN channel (i.e., one without fading, for which $A \equiv 1$), but it also represents the average SNR value on the fading channel, taking into account that $\mathrm{E}\{A^2\} = 1$.

4.5.3 The binary-input Gaussian channel (BIAWGN)

We now come back to a model of a BPSK link disturbed by AWGN in which we have a *binary* modulation, and consider as the channel output the soft value of the sampled decision variable $z[n]$ at the output of the matched filter (Binary-Input AWGN channel, BIAWGN). We also add a binary channel encoder with coding rate r as shown in Fig. 4.18. The channel is again a discrete-time *additive* Gaussian channel, but of a different nature from all those considered so far: it is a *binary input* and *continuous output* (soft) channel. The channel

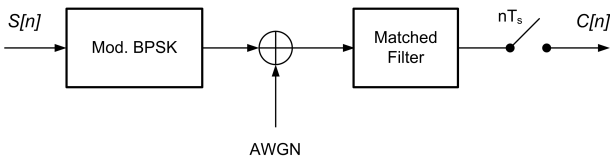


Figure 4.18 Binary-input Gaussian channel (BIAWGN)

capacity can again be obtained as (it is an additive channel)

$$c_{BIAWGN} = \max_{f_S} [h(C) - h(C|S)] = \max_{f_S} h(C) - \log\left(\sqrt{2\pi e \sigma_W^2}\right) \tag{4.89}$$

where σ_W^2 is the variance of the channel noise component. If we use root-Nyquist shaping for the transmit and receive filter, and we normalize the amplitude of the transmitted symbols to the value ± 1, we easily find that (see (2.92))

$$\sigma_W^2 = (2E_s/N_0)^{-1} = (2rE_b/N_0)^{-1} \tag{4.90}$$

To calculate $h(C)$, we must now derive $f_C(c)$, and then proceed to maximize it with respect to the probability of the source symbols. We can assume, given the symmetry of the probability density of the noise and of the modulation levels, that maximization of $h(C)$ with respect to the distribution of the source symbols occurs when they are equally probable, so

$$f_C(c) = \frac{1}{2} f_C(c|0) + \frac{1}{2} f_C(c|1) = \frac{1}{2} \frac{1}{\sqrt{2\pi\sigma_W^2}} \left[\exp\left(-\frac{(c+1)^2}{2\sigma_W^2} \right) + \exp\left(-\frac{(c-1)^2}{2\sigma_W^2} \right) \right]$$

$$= \frac{1}{2} \sqrt{\frac{E_s/N_0}{\pi}} \left[\exp\left(-\frac{E_s}{N_0}(c+1)^2 \right) + \exp\left(-\frac{E_s}{N_0}(c-1)^2 \right) \right] \tag{4.91}$$

so that

$$c_{BIAWGN} = -\int_{-\infty}^{\infty} \frac{1}{2} \sqrt{\frac{E_s/N_0}{\pi}} \left[\exp\left(-\frac{E_s}{N_0}(c+1)^2 \right) + \exp\left(-\frac{E_s}{N_0}(c-1)^2 \right) \right]$$

$$\cdot \log\left(\frac{1}{2} \sqrt{\frac{E_s/N_0}{\pi}} \left[\exp\left(-\frac{E_s}{N_0}(c+1)^2 \right) + \exp\left(-\frac{E_s}{N_0}(c-1)^2 \right) \right] \right) dc$$

$$-\frac{1}{2} \log\left(\frac{\pi e}{E_s/N_0} \right) \quad bit/c.u. \tag{4.92}$$

This capacity, which can never be greater than 1, can be computed numerically, and is shown in Fig. 4.19. We notice the improvement compared to the already known BPSK/BSC channel with matched filter, in which the soft output is predetermined hard-detected *before channel decoding*. As already anticipated, the difference for intermediate E_s/N_0 ratios is about 2 dB for the same capacity.

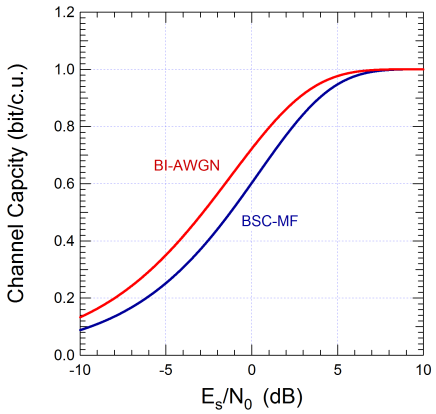


Figure 4.19 Comparison of BIAWGN and BPSK/BSC capacities

4.5.4 The complex-valued AWGN channel

We have seen that the BIAWGN channel is intrinsically limited to a maximum capacity of 1 bit/c.u. because the input is forced to be binary. The same reasoning must be applied when calculating the capacity of a modem operating on AWGN, and for which the signal format (i.e., the constellation) is established in advance (QPSK, 16-QAM, etc.). That is, we need to obtain capacity curves similar to (5.72), for a channel with a digital input equal to a particular constellation, and a soft output derived from the AWGN. The calculation is very similar to that of (5.72), but we must take into account that the capacity must be evaluated on the *baseband equivalent* of the channel, that is, reasoning on a complex I/Q signal.

Without any constraints on the input constellation, the calculation of the unconstrained *complex-valued* SISO (Soft-Input, Soft-Output) discrete-time AWGN channel is simple:

$$\mathcal{C}_{I/Q} = \log(1 + \text{SNR}) \quad \text{(bit/simbolo)} \tag{4.93}$$

a value that is *double* compared to (4.84). The result is very easily explained: the two I/Q components of the complex signal represent *two* independent communication channels (because the I/Q components of the AWGN are independent), and therefore the total capacity is the *sum* the capacities of the two *parallel channels* capacitances.

How can we try to attain such capacity? In practice, the complex AWGN channel is used by choosing an appropriate COD/MOD format, i.e., an error-protection channel coding algorithm followed by an appropriate I/Q modulation. The channel conveys I/Q symbols belonging to a constellation of M points, onto which the encoded binary symbols obtained from a binary code with rate r are appropriately mapped (for example, using Bit-Interleaved Coded Modulation (BICM) technology). In this case, the I/Q symbols each carry an amount of information equal to $r \cdot \log(M)$, and the reliable communication condition becomes

$$r \cdot \log(M) \leq \mathcal{C}_{I/Q} \quad \rightarrow \quad M^r \leq 1 + \text{SNR} \tag{4.94}$$

If, as often happens, SNR $\gg 1$ and expressing SNR in dB, we have

$$r \cdot N_b \leq \frac{\text{SNR}(dB)}{3} \tag{4.95}$$

where N_b is the number of encoded binary symbols associated with each constellation point.

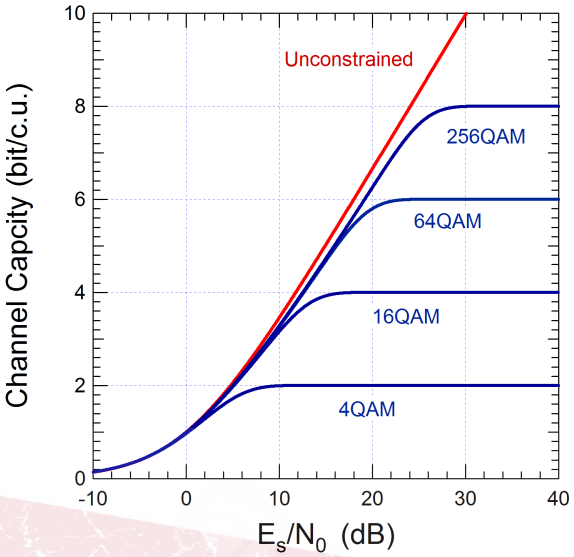


Figure 4.20 Capacity of different I/Q constellations on the AWGN

In the case of a complex AWGN channel with a *digital* I/Q input, i.e., with a preassigned constellation, the capacity calculation is similar to that described in (5.72), but the integration to calculate the various differential entropies must generally be performed on the complex plane because $f_C(c)$ is an equiprobable two-dimensional Gaussian mixture with conditional means equal to the various points of the I/Q constellation. The result, however, is simple, closely resembling the curve in Fig. 5.40, and is shown in Fig. 4.20. As with the BIAWGN, the capacity does not grow indefinitely but "saturates" at the value $\log(M)$ where M is the number of symbols in the constellation.

4.5.5 The continuous-time (waveform) Gaussian channel

We are now to generalize the capacity formula for the discrete-time AWGN channel. If the signaling rate on the channel is $R_s = 1/T_s$ c.u./s, the capacity in terms of bits/s is

$$\mathcal{C}_{AWGN} = R_s \cdot c_{AWGN} = \frac{1}{2T_s} \log\left(1 + \frac{\sigma_S^2}{\sigma_W^2}\right) \quad \text{(bit/s)} \tag{4.96}$$

and the reliable communication condition is

$$R_b = R_s \cdot N_b \leq R_s \cdot c_{AWGN} = \mathcal{C}_{AWGN} \tag{4.97}$$

where N_b is the number of bits per symbol of the COD/MOD method that is used.

We can now extend this formula to the so-called AWGN *waveform channel* represented in Fig. 4.21. How can we extend the result (4.83) to our case, that is, move from a discrete-time context to an equivalent continuous-time one? We must assume that the two cases have a sampling/interpolation relationship according to the relations (2.61)-(2.78), and also that, to make the two contexts completely equivalent, all the continuous-time signals that are involved are *band-limited* to B, and sampled with a sampling frequency $f_s = 2B$. In particular, the AWGN will also be band-limited, that is, characterized by a power spectral

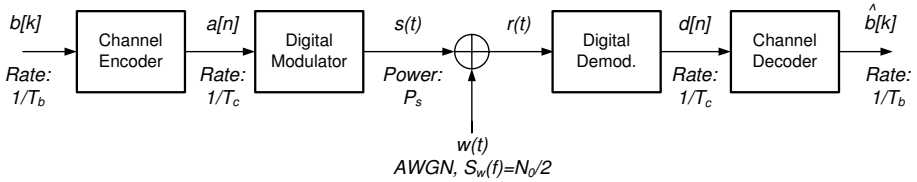


Figure 4.21 Bandlimited continuous-time AWGN channel (waveform channel)

density $S_W(f) = N_0/2$ that is constant within the signal band B and 0 outside, and will therefore have by a power $P_W = \sigma_W^2 = N_0 B$. As for the signal, we know that capacity is achieved when the source is Gaussian (and of course band-limited). In this case, it can be represented, according to the sampling theorem, by $2B$ samples per second, all of which will have the same power $P_S = \sigma_S^2$. Under such hypothesis, the waveform channel is equivalent to a discrete-time channel in which the signaling rate (channel usage rate) is identified with the signal sampling frequency: $f_s = 2B = 1/T_s$. Applying the formula already found for discrete time, we conclude

$$\mathcal{C}_{AWGN} = \frac{1}{2T_s} \log\left(1 + \frac{\sigma_S^2}{\sigma_W^2}\right) = B \log\left(1 + \frac{P_S}{N_0 B}\right) = B \log\left(1 + SNR\right) \quad \text{(bit/s)} \tag{4.98}$$

This is Shannon's fundamental result on the capacity of the bandlimited AWGN channel, a benchmark to the performance of practical digital links. Since the reliability condition is

$$R_b \leq B \log\left(1 + SNR\right) \simeq B \, \frac{SNR_{dB}}{3} \tag{4.99}$$

we see that the reliable bit rate is directly proportional to the available bandwidth (and this is not surprising), but also depends through a further "boost" factor on the quality of link: it is also proportional to the logarithm of the signal-to-noise ratio (assuming the latter is $\gg 1$), or, in practice, to the signal-to-noise ratio expressed in dB.

4.5.6 Source-Channel Separation

In closing this chapter on channel coding, and reconsidering the source coding algorithms of the previous chapter, something seems to be missing between the two topics. The need to introduce a certain amount of redundancy to construct a channel code and make the data robust may seem to conflict with the need, much emphasized in the previous chapter, to remove as much source redundancy as possible to make the data efficient. A perfectly legitimate question is the following: before completely removing source redundancy, wouldn't it be more expedient to remove a *controlled* amount of it, so as to leave a "fair" residue that adapts the source to the noisy channel? This is equivalent to saying that one should/could perform joint source/channel coding - in the final analysis, what is required for reliable transmission is $H(S) \leq c$.

The answer to the previous question is *NO, it is not*. Aside from the practical difficulty of developing algorithms that are optimal both with respect to the source and to the channel, in the simple case that we have discussed so far, i.e., memoryless source and channel, there is no need for joint encoding: we can first (via the data compression theorem) represent the source information with zero redundancy, and then, having done this, we can reliably communicate the compressed bits over a noisy channel with an appropriate channel code. In

other words, optimal joint source/channel coding is... *disjoint* coding. This property holds even for more complicated and/or realistic cases, i.e., sources and channels with memory: it is always possible to achieve efficient and robust communication over a noisy channel by using separate encodings. Although expressed colloquially, this is a *third* Shannon theorem called the "Source-Channel Separation Theorem".

CHAPTER 5

CHANNEL COEING CHAVNEL CODING CHANNEL CODING

"Repeating Helps"

—Latin Motto

This famous Latin sentence, of uncertain origin, is the basis of the simplest form of channel coding, namely, the repetition code.

5.1 Shannon Capacity and Fundamentals of Channel Coding

The purpose of this chapter is to discuss the main coding schemes whose performance lies close as possible to the Shannon limit (5.69). The channel coding theorem seen in 4.4.3 unfortunately does not tell us how to find these codes; however, almost 80 years of research in this field culminated in the discovery (or rediscovery) in the 1990s of families of codes with iterative decoding like turbo and LDPC codes and, more recently (2010), the polar codes whose performance comes very close to the limit. We will retrace the historical development of the technology, starting with linear block codes, continuing with convolutional codes, then introducing turbo/LDPC codes, and finishing with polar codes.

5.1.1 Block Codes and Optimum Decoding

In general, an (N, K) binary block code with rate $r = K/N$ is defined by two ingredients. First, we have to identify a set of 2^K strings or *codewords* $\mathbf{a}_i$, $i = 0, 1, ..., 2^K - 1$ made of N binary coded symbols each. This set is also called the *dictionary* of the code (or *codebook*), and is clearly a judiciously selected subset of all of the 2^N possible binary strings of length N. Second, we have also to specify the *mapping rule* that is used by the channel encoder to map each of the 2^K possible source bit strings with length K (blocks) $\mathbf{b}_i$, $i = 0, 1, ..., 2^K - 1$, onto the pre-selected codewords. This is what we have done in the definition of the simple repetition code. After encoding is specified in such a way, the codewords are sent onto a noisy channel and can be corrupted by noise. The channel decoder takes such noisy inputs and does his best to find that codeword in the codebook that is the most likely having been sent, and finds the source bits associated to that - in a word, *decodes* the source block.

Decoding of the simple $(N, 1)$ repetition code of the previous subsection was easy, and has also a nice *geometrical* interpretation that tells us more bout how to design good codes (and good decoders). Assume we're using $N = 3$ repetitions only, and that we send out our coded symbols on a noisy channel. After matched filtering and regeneration, what we get is a word of three binary symbols that may take any value in a set of $2^N = 8$ elements. We know in advance that not all of these received words belong to the so-called *dictionary* of the code (or *codebook*), that in our case is made of two allowed words only: 000 and 111. We can depict this situation as in Fig. 5.1 where we see an $N-$dimensional space whose points represent $N-$dimensional binary words. The large dots are the words belonging to the codebook, the other points are words that can be received but that contain errors. Just to make an example, assume that an information bit equal to 1 is sent, and that an error on the first coded symbol is produced, so that the received word is 011. The majority decoder will decode correctly, just because amidst all of the possible words in the code dictionary, we see form Fig. 5.1 that 111 (corresponding to the transmission of 1) is the *closest* to what has been received. On the contrary, if *two* channel errors are produced so that we get, say, 010, we see that the closest word in the codebook is now 000, and we have a decoding error.

Can we formalize this geometric interpretation? Let us call $\mathbf{b} = [b[0], b[1], \ldots, b[K-1]]$ the information block, $\mathbf{a} = [a[0], a[1], \ldots, b[N-1]]$ the code block, and $\mathbf{d} = [d[0], d[1], \ldots, d[N-1]]$ the block of binary symbols received form the channel, with possible errors. What a decoder has to do is clear: observing the output of the channel $\mathbf{d}$ (that is, a block of hard-detected coded symbols), and finding that particular codeword $\hat{\mathbf{a}}$ amidst those in the codebook Γ whose a-posteriori probability $p(\hat{\mathbf{a}}|\mathbf{d})$ is maximum (Maximum a-posteriori probability rule):

$$\hat{\mathbf{a}} = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmax}}\, p(\tilde{\mathbf{a}}|\mathbf{d}) \tag{5.1}$$

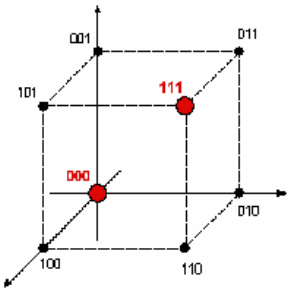


Figure 5.1 Geometric representation of the repetition code with $N = 3$

The decoded source block $\hat{\mathbf{b}}$ is then inverse-mapped from $\hat{\mathbf{a}}$. Using Bayes' rule, $p(\tilde{\mathbf{a}}|\mathbf{d}) = p(\tilde{\mathbf{d}}|\mathbf{a})p(\mathbf{a})/p(\mathbf{d})$, but $p(\mathbf{d})$ does not depend on $\tilde{\mathbf{a}}$, and also all $\tilde{\mathbf{a}}$ are equiprobable (since all source bits are equiprobable and independent to each other). So our original MAP problem (5.1) is turned into the equivalent *maximum-likelihood* one:

$$\hat{\mathbf{a}} = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmax}}\, p(\mathbf{d}|\tilde{\mathbf{a}}) \tag{5.2}$$

or, equivalently

$$\hat{\mathbf{a}} = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmax}}\, \ln p(\mathbf{d}|\tilde{\mathbf{a}}) \tag{5.3}$$

Since the errors on the channel are independent of each other, we also have

$$p(\mathbf{d}|\tilde{\mathbf{a}}) = p(d[0]|\tilde{a}[0]) \cdot p(d[0]|\tilde{a}[0]) \cdot \ldots \cdot p(d[N-1]|\tilde{a}[N-1]) \tag{5.4}$$

Recall now that $\mathbf{d}$ is a given vector, as observed at the output of the hard-detected channel, whilst $\tilde{\mathbf{a}}$ is the one that varies across the codebook. So the likelihood (5.4) is easy to find: it just amounts to

$$p(\mathbf{d}|\tilde{\mathbf{a}}) = (p_e)^{N_{eq}} \cdot (1-p_e)^{N_{neq}} \tag{5.5}$$

where p_e is the probability of a single channel error, and where we used a funny notation: N_{eq} indicates "the number of entries of $\tilde{\mathbf{a}}$ that are equal to the corresponding entries of $\mathbf{d}$", whilst N_{neq} is "the number of entries of $\tilde{\mathbf{a}}$ that are different from the corresponding entries of $\mathbf{d}$". Of course, $N_{neq} = N - N_{eq}$. Using this,

$$\ln p(\mathbf{d}|\tilde{\mathbf{a}}) = N_{neq} \ln p_e + (N - N_{neq}) \ln(1-p_e) \tag{5.6}$$

The quantity N_{neq} indicating the number of positions where two binary vector differ is actually called the *Hamming distance* $d_H(\mathbf{d}, \tilde{\mathbf{a}})$ between $\mathbf{d}$ and $\tilde{\mathbf{a}}$. [1] Considering this,

$$\ln p(\mathbf{d}|\tilde{\mathbf{a}}) = -d_H(\mathbf{d}, \tilde{\mathbf{a}}) \ln \frac{1-p_e}{p_e} + N \ln(1-p_e) \tag{5.7}$$

The second term in (5.7) does not depend on $\tilde{\mathbf{a}}$, so that, whatever the probability of a channel error is, our final ML problem can be re-cast into

$$\hat{\mathbf{a}} = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmax}}\, p(\mathbf{d}|\tilde{\mathbf{a}}) = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmin}}\, d_H(\mathbf{d}, \tilde{\mathbf{a}}) = \tag{5.8}$$

[1] The Hamming distance of a generic vector from the null vector, i.e., the number of 1s in the vector is also called the *Hamming weight* of the vector

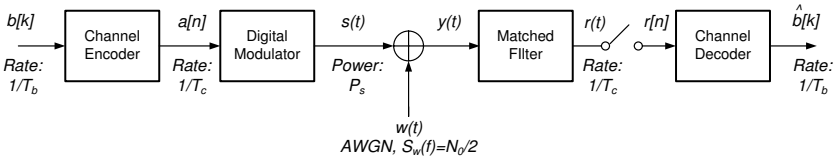


Figure 5.2 Soft-input decoding of a code

The decoder has to find, amidst all codewords, the one whose Hamming distance from the received word **d** is the minimum - juts like the majority decoder of a repetition code. Our naive geometric interpretation is fully confirmed by the theoretical analysis of the optimum detector.

This "find the closest" rule is the optimum decoding criterion also assuming an AWGN channel with independent noise samples on the coded symbols:

$$r[n] = a_{\pm}[n] + w[n] \quad \Rightarrow \mathbf{r} = \mathbf{a}_{\pm} + \mathbf{w} \tag{5.9}$$

where $a_{\pm}[n]$ is the re-mapped version of $a[n]$ on the symmetric alphabet ± 1. In the example of the repetition decoder, we loosely spoke of "distance" alluding both to the *Hamming distance* we now know about, but also to the conventional *Euclidean* distance of Euclidean geometry, since in the space of Fig. 5.1 two Euclidean-close points are also Hamming-close. With *hard-decoding* on the channel, the only distance that makes sense is Hamming's. But, if we have an AWGN channel with matched-filter decoding, we can also do *soft-decoding* if we directly take the continuous-amplitude signal samples at the output of the matched filter without hard decision detection, as shown in Fig. 5.2. In this case, the $N-$dimensional point $\mathbf{r} = [r[0], r[1], ..., r[N-1]]$ representative of the received signal can lie *anywhere* in R^N, not necessarily on one of the 2^N possible binary words. Nonetheless, the decoder still has to find the allowed codeword that is the closest to what has been received, that is, that codeword that minimizes the Euclidean distance

$$d_E(\mathbf{r}, \tilde{\mathbf{a}}) \stackrel{\triangle}{=} \|\mathbf{r} - \tilde{\mathbf{a}}\| = \sum_{n=0}^{N-1} (r[n] - \tilde{a}[n])^2 \tag{5.10}$$

Example 5.26

Let us show that minimization of the Euclidean distance (5.10) leads to maximum-likelihood estimation of the transmitted codeword when the observed signal is "soft-detected" as in Fig. 5.2. The estimation problem is now, skipping some details,

$$\hat{\mathbf{a}} = \underset{\tilde{\mathbf{a}} \in \Gamma}{\operatorname{argmax}} \ln f_{\mathbf{R}}(\mathbf{r}|\tilde{\mathbf{a}}) \tag{5.11}$$

where $f_{\mathbf{R}}(\mathbf{r}|\tilde{\mathbf{a}})$ is the conditional pdf of the received "soft" vector $\mathbf{r}$. On the other hand, the samples $r[n]$ at the output of the matched filter (conditioned on a specific value of the trial coded symbol $\tilde{a}[n]$) are given by $r[n] = \tilde{a}[n] + w[n]$ where $w[n]$ is a sequence of independent Gaussian random variables with zero mean and a variance σ^2. The joint pdf of $\mathbf{r}$ is therefore Gaussian multivariate with uncorrelated components:

$$f_{\mathbf{R}}(\mathbf{r}|\tilde{\mathbf{a}}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \prod_{n=0}^{N-1} \exp\left\{-\frac{(r[n] - \tilde{a}[n])^2}{2\sigma^2}\right\} \tag{5.12}$$

and its logarithm is

$$\ln f_{\mathbf{R}}(\mathbf{r}|\tilde{\mathbf{a}}) = C - \frac{1}{2\sigma^2}\sum_{n=0}^{N-1}(r[n]-\tilde{a}[n])^2 \tag{5.13}$$

where $C = -N/2\ln(2\pi\sigma^2)$ does not depend on $\tilde{\mathbf{a}}$. We see that maximization of $\ln f_{\mathbf{R}}(\mathbf{r}|\tilde{\mathbf{a}})$ is accomplished by minimization of the Euclidean distance (5.10), irrespective of the particular value of the noise variance.

Next question is: what is the "best" code for a certain codewords length? Our discussion on the decoding criterion suggest an answer to this question: the one that has most "scattered" codewords, so that it is most difficult to take wrong decisions. Coming back to Fig. 5.1, we see that 000 an 111 are actually the most scattered amidst all possible words with $N = 1$. But another good code with equal performance would also be, for instance, 011 and 100 (the decoder would be more complicated than simple majority, though). What actually matters in the design of the code is the *minimum distance* between any two codewords. It is this minimum distance that drives the BER of decoding, since it determines the frequency of error events. If the minimum distance of the code is relatively large, confusing a codeword with another one (its closest one in the codebook) requests a larger amount of noise than with a code whose minimum distance is smaller.

Example 5.27

Can we find the optimum decoder for the repetition code on the soft-output AWGN channel as in the previous example? What we have to do is deriving the decision rule that minimizes the Euclidean distance (5.10) of the received vector $\mathbf{r}$ with respect to the two possible codewords 00...00 and 11...11. Assuming that the source symbol 0 is mapped into -1, and 1 is mapped into +1, the decision rule is

$$\sum_{n=0}^{N-1}(r[n]-(-1))^2 \underset{\hat{b}[0]=1}{\overset{\hat{b}[0]=0}{\lessgtr}} \sum_{n=0}^{N-1}(r[n]-(+1))^2 \tag{5.14}$$

Expanding the squares, and eliminating the common terms appearing at both sides, we get

$$\sum_{n=0}^{N-1} r[n] \underset{\hat{b}[0]=1}{\overset{\hat{b}[0]=0}{\lessgtr}} 0 \tag{5.15}$$

The decoder combines all the soft inputs $r[n]$ pertaining to a code block (and this can be interpreted as *smoothing* the effect of noise), and performs a final threshold decision with respect to zero on the resulting variable. The reader can show that the coding gain of the repetition code with this kind of maximum-likelihood soft-input decoding on the AWGN channel is equal to N.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array} \quad (a) \qquad \begin{array}{ccccc} 0 & 1 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 1 & 0 & \mathbf{1} \\ 1 & 1 & 1 & 0 & \mathbf{1} \\ 1 & 0 & 0 & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \end{array} \quad (b)$$

Figure 5.3 Example of a double parity check code

Where is the issue that motivates the large number of different codes that are used in the practice? The real issue lies in *decoding*. It may be relatively easy to find good codes (that is, good *codebooks*) characterized by good distance properties. But, a different thing is finding good *decoding algorithms* for good codes that can be implemented with reasonable complexity. In fact, a straightforward ML decoder that computes all 2^K possible distances from the received signal $\mathbf{r}$ to each allowed codeword $\mathbf{a} \in \Gamma$, and finds the minimum one is impractical as soon as K is larger than, say, a few tens impractical. So designing a code is actually designing a codebook with reasonably good distance properties, but especially with other additional properties of some kind that allow for easy *decoding*. Historically, the first samples of such codes are parity check-codes and Hamming codes, introduced back in the 50's, that are representative of the family of *algebraic linear block codes*. They heavily rely on the theory of vector spaces and/or Galois fields, whose flavor we try to give in the following

5.1.2 Algebraic Block Codes

5.1.2.1 The double parity-check code From the discussion in the previous paragraph, we know that a $K = 1$, $N = 3$ repetition code with rate $r = 1/3$, can correct up to a single channel error in a block of 3 channel outputs. This can be even "too good" in the sense that errors on the channel can be much more sporadic than 1 in three symbols, whilst at the same time a coding rate of 1/3 may reveal, after all, too small (too much added redundancy). So we may wonder how to build another simple code that corrects a single error on a longer source block, still meeting the specifications on the channel BER, but with a larger coding rate (less redundancy) than the repetition code. A simple example is the *double parity check* code (the so called *product code* invented by Elias in the 50's) that is still used at times. Assume that the source block has length $K = M^2$, so that we can arrange the source bit in a square table as in Fig. 5.3 (a), where $K = 16$. The encoder adds to the table the horizontal (vertical) parity checks on the right (on the bottom). Each check is obtained by modulo-2 adding the bits on each row (column) so that the resulting new row (column) has an *even* number of 1s. We end up with the new table in Fig. 5.3 (b) that represents the codeword to be sent out on the channel. The source bits are sent out with their previous, natural order, and the checks are sent afterwards in any convenient order so that the resulting code is *systematic* (i.e., the source bits appear unchanged in the codeword), and the blocklength is $N = M^2 + 2M = 24$, The coding rate is $r = M^2/(M^2 + 2M)$ that is very close to 1 when M (K) is large. In our example, $r = 2/3$. The error correction mechanism of such a code is very simple and neat. The decoder reconstructs a table similar to the one in Fig. 5.3 (a) starting from the hard-detected received codeword $\mathbf{d}$. In particular, it re-computes the parities based on the information-bearing, systematic first section of $\mathbf{d}$ (M^2 symbols) and

0	1	1	1	0		0	1	1	1	0	
0	0	1	0	0		0	0	1	0	**1**	
1	**0**	1	0	**1**	(a)	1	1	1	0	0	(b)
1	0	0	1	0		1	0	0	1	0	
0	**1**	0	0			0	0	0	0		

Figure 5.4 Error correction in a double parity check code

adds them to the hard-detected parity symbols contained in the second section of $\mathbf{d}$ itself ($2M$ symbols). The resulting new symbols (re-computed parities plus received parities) are called the *syndrome* symbols. If there is no channel error on the block, the re-computed parities are equal to the received parity symbols, and the syndrome symbols are all equal to 0. In this case, no action is performed and the received information bits are supposed to be correct. But assume now that the channel has produced a single error on a source bit, like in Fig. 5.4) (a) (bold): some syndrome symbols are different from zero, since the re-computed parities that include the error are different form the received ones (Fig. 5.4) (a) again). The syndrome symbols equal to 1 mark the "coordinates" of the wrong bit that can be identified and corrected. If there is a single error on a parity symbol as in Fig. 5.4) (b), then there is a *single* syndrome symbol equal to 1, and the decoder knows that the error has happened on a parity bit, but also that the source bits are indeed correct: no further action is performed.

5.1.2.2 Definition of algebraic linear code Can we generalize such a construction? What are the tools than can lead to the design of good and simple codes such the double-parity check? We start by introducing the binary algebraic *field* $\Omega_2 \triangleq (\{0,1\},+,\cdot)$ that is, the set of binary digits with associated the modulo-2 operations of addition and product. The introduction of Ω_2 also induces the definition of the $N-$dimensional (finite) *vector space* Ω_2^N: the elements of Ω_2^N are the 2^N $N-$tuple $\mathbf{v} = [v[0], v[1], ..., v[N-1]]$, $v[n] \in \Omega_2$, that we will consider as row vectors. A linear (N,K) block code $(K < N)$ is a $K-$dimensional vector subspace Γ (the codebook or dictionary) of Ω_2^N containing 2^K elements (the codewords). Denoting as usual with $\mathbf{b} = [b[0], b[1], ..., b[K-1]]$ a word (block) of K information bit symbols, we associate to $\mathbf{b}$ a codeword (block) $\mathbf{a} = [a[0], a[1], ..., a[N-1]] \in \Gamma$. The rate of the code is clearly $r = K/N$, and fixed-length coded symbols blocks $\mathbf{a}$ correspond to fixed-length information bit blocks $\mathbf{b}$. Why do we say that the codebook Γ is a vector subspace of Ω_2^N? Because we can identify K linearly independent elements $\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_{K-1} \in \Omega_2^N$ that are a *linear base* of Γ . This means that every element (codeword) of Γ can be decomposed as a linear combination of the elements of the base. In particular, we can exploit this property to easily set a map between the information blocks $\mathbf{b}$ and the codewords $\mathbf{a}$:

$$\mathbf{a} = b[0] \cdot \mathbf{g}_0 + b[1] \cdot \mathbf{g}_1 + ... + b[K-1] \cdot \mathbf{g}_{K-1} \tag{5.16}$$

Since $\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_{K-1}$ is a base of the subspace, different information blocks generate different codewords, and the code is uniquely *decodable* in the absence of any channel error. The "encoding" relation (5.16) can be summarized as follows:

$$\mathbf{a} = \mathbf{b} \cdot \mathbf{G}_{ns} \tag{5.17}$$

where we introduced the *generator matrix*

$$\mathbf{G}_{ns} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ ... \\ \mathbf{g}_{K-1} \end{bmatrix}_{K \times N} \tag{5.18}$$

Note that the *rank* of the (rectangular) matrix $\mathbf{G}_{ns}$ is for sure K since the rows of it are a base of the subspace Γ and so they are linearly independent [2]. Creating the vector subspace means selecting within Ω_2^N those 2^K codewords, or *points* in the space that have to be maximally separated to yield a good code.

Before speaking of distance and decoding, we observe that the code defined by (5.17), and in particular by $\mathbf{G}_{ns}$ is nonsystematic (hence the subscripts). But we can attain to an equivalent systematic code very easily using *Gauss-Jordan* elimination, so as to re-shape $\mathbf{G}_{ns}$ into an equivalent matrix $\mathbf{G}$ as follows:

$$\mathbf{G} = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \cdots & 0 & \bullet & \bullet & \cdots & \bullet \\ 0 & 1 & 0 & \cdots & 0 & \bullet & \bullet & \cdots & \bullet \\ 0 & 0 & \ddots & \ddots & \vdots & \bullet & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & \bullet \\ 0 & 0 & \cdots & 0 & 1 & \bullet & \cdots & \bullet & \bullet \end{array}\right]_{K \times N} = \left[\mathbf{I}_{K\times K} | \mathbf{P}_{K\times(N-K)}\right] \tag{5.19}$$

where $\mathbf{P}$ is the *parity-generation* matrix. Of course, the specific mapping (correspondence) of information blocks to codewords changes (in particular the Gauss-Jordan method requires permutation of rows and/or columns), but the code Γ as a whole, and in particular its distance properties, stays the same, since the subspace has the same base and hence it *is* the same. Using this new generator matrix $\mathbf{G}$ guarantees that the code is indeed systematic since the first K bits in the codeword are just a replica of the K information bits. The additional $N-K$ bits that are appended to the information block are generated by the matrix $\mathbf{P}$ and are the *parity* bits that give redundance to the codeword.

Example 5.28

A trivial example is again the repetition code. For this kind of coding, $K = 1$, and the generator matrix is clearly the $1 \times N$ matrix $\mathbf{G} = \mathbf{g}_0 = \mathbf{1}_N$, that is, the N-dimensional row vector made of all 1s. The generator matrix is already systematic, and the parity-generation matrix is $\mathbf{P} = \mathbf{1}_{N-1}$.

5.1.2.3 Parity-check matrix If we collect now all the elements of Ω_2^N that are *not* into Γ we get the *orthogonal subspace* of Γ that we denote with $\Gamma^\perp$ and whose dimensionality is of course $N-K$. We can find a linear base for this subspace made of the $N-K$ vectors $\mathbf{h}_0, \mathbf{h}_1, ..., \mathbf{h}_{N-K-1}$. If we take any codeword $\mathbf{a}$ in Γ, we have

$$\mathbf{a} \cdot \mathbf{h}_i = 0 \ , \ \ i = 0, 1, ..., N-K-1 \tag{5.20}$$

[2] the rank of a matrix is just the maximum number of rows/columns that turn out to to be linearly independent

This is easy to understand since any codeword in Γ is orthogonal to any element in $\Gamma^{\perp}$, and this can hold only iff a codeword is orthogonal to *all* of the elements in the base set of $\Gamma^{\perp}$ itself. Of course, if we take at random a vector $\mathbf{d}$ in Ω_2^N, it may happen that $\mathbf{d} \cdot \mathbf{h}_i \neq 0$ as long as $\mathbf{d} \notin \Gamma$. We can always decompose $\mathbf{d}$ as the sum of two components, a codeword $\mathbf{a} \in \Gamma$ plus an orthogonal term $\mathbf{e} \in \Gamma^{\perp}$, so that $\mathbf{d} = \mathbf{a} + \mathbf{e}$ with $\mathbf{a} \cdot \mathbf{h}_i = 0$ e $\mathbf{e} \cdot \mathbf{h}_i \neq 0$.

The $N - K$ equations (5.20) are called *parity checks* since they generalize the notion of parity-control symbol that we have already introduced for the product code. We can resume the parity checks into a single matrix relations:

$$\mathbf{a}\mathbf{H}^T = \mathbf{0}_{N-K} \tag{5.21}$$

where we have introduced the parity-check matrix

$$\mathbf{H} \stackrel{\triangle}{=} \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \cdots \\ \mathbf{h}_{N-K-1} \end{bmatrix}_{(N-K)\times N} \tag{5.22}$$

The parity check matrix can be used to perform *error control* on the block $\mathbf{d}$ of hard-detected binary symbols at the output of a noisy channel. If no errors are produced, $\mathbf{d} = \mathbf{a}$ (for that particular $\mathbf{a} \in \Gamma$ that was sent on the channel) and so

$$\mathbf{d}\mathbf{H}^T = \mathbf{0} \tag{5.23}$$

If some of the parity equations fail, we know for sure that errors were introduced by the channel, so that we accomplish the function of *error detection*.

Until now, nothing has been said about how to derive $\mathbf{H}$. The funny thing is that $\mathbf{H}$ comes for free if we can cast our generator matrix into a systematic form. We can show in fact that

$$\mathbf{H}_{(N-K)\times N} = \left[\mathbf{P}^T_{(N-K)\times K} | \mathbf{I}_{(N-K)\times(N-K)}\right] \tag{5.24}$$

Example 5.29

What is the appearance of the parity check matrix of the repetition code? Using (5.24) we find

$$\mathbf{H}_{(N-1)\times N} = \left[\mathbf{1}^T_{(N-1)\times 1} | \mathbf{I}_{(N-1)\times(N-1)}\right] \tag{5.25}$$

or, for the particular case $N = 5$,

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad , \quad \mathbf{H}^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.26}$$

and the meaning of the $N - 1$ parity equations is clear: each coded symbol $a[n]$ with $n > 0$ has to be equal to $a[0]$.

Before tackling the issue of decoding, we want to say something more about the relation between the distance properties of the code and its error-correcting capability. The good-old repetition code has a minimum distance $d_{min} = N$ and can correct up to $(N-1)/2$ error (N odd). The nice thing is that this property holds for *any* code: the key parameter driving the error correction capability is the *minimum distance* between two any codewords

$$d_{min} \stackrel{\triangle}{=} \min_{\mathbf{a}_i, \mathbf{a}_k \in \Gamma} d_H(\mathbf{a}_i, \mathbf{a}_k) \tag{5.27}$$

Take for instance the celebrated (7,4) systematic Hamming code

$$\mathbf{G}_{K\times N} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}_{4\times 7} \tag{5.28}$$

with rate $r = 4/7$ and parity-check matrix

$$\mathbf{H}^T_{N\times(N-K)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{7\times 3} \tag{5.29}$$

has a minimum distance equal to 3, and corrects up to 1 channel error with a relatively low redundancy.

The reader may prove that the minimum distance d_{min} of a linear algebraic block code ia (also) the minimum distance of any codeword from the null vector $\mathbf{0}_N$, therefore it is (also) the minimum Hamming weight of any codeword. In addition, it is (also) equal to the minimum number of rows of the parity-check matrix $\mathbf{H^T}$ that need to be added to give the null vector $\mathbf{0}$. Hamming linear block codes, invented by the coding pioneer Richard Hamming in the 50's, were the first examples of highly-structured codes with better distance properties than many of the previously ad-hoc designed error-correcting codes.

5.1.2.4 Syndrome decoder How can we decode Hamming codes (and linear block codes in general)? The simplest way of detecting errors is using the so-called *syndrome decoding* technique. To understand what we are talking about, we introduce the notion of *error vector* or *error pattern*. In the presence of channel errors we know that in general $\mathbf{d} \neq \mathbf{a}$, so that we can write

$$\mathbf{d} = \mathbf{a} \oplus \mathbf{e} \tag{5.30}$$

where $\mathbf{e}$ is an N-dimensional vector that "marks" the position of errors that were introduced by the channel on the codeword $\mathbf{a}$. As mentioned before, we can re-check the parity of the received codeword by using the parity-check matrix:

$$\mathbf{d}\mathbf{H}^T = (\mathbf{a} + \mathbf{e})\mathbf{H}^T = \mathbf{e}\mathbf{H}^T \stackrel{\triangle}{=} \mathbf{s} \tag{5.31}$$

The $(N-K)$-dimensional vector $\mathbf{s}$ that is not null in the presence of channel errors is called the *syndrome* of the received block, and is the starting point for error detection. It is seen from (5.31) that the syndrome does *not* depend of the particular codeword that was sent. Rather, it is a function of the particular error pattern introduced by the channel, and "marks" the particular parity check equation that fails due to errors. Unfortunately, it does not directly mark channel errors, and this is easy to understand: we may have up to 2^N different error patterns, whilst we only have 2^{N-K} different syndromes, so that different error vectors may give in general the same syndrome.

So what is the next step for decoding? In principle, we could do like this. We start by listing in a column all 2^K possible N-bit codewords $\mathbf{a}_0, ..., \mathbf{a}_{2^K-1}$ starting with $\mathbf{a}_0 = \mathbf{0}$. We regard this column as the first of a matrix $\mathbf{A}$ containing *all possible* 2^N *channel blocks*. The first *row* of the matrix starts of course with $\mathbf{a}_0$, that this time is interpreted as the *all-zero error pattern*, i.e., $\mathbf{a}_0 = \mathbf{0} = \mathbf{e}_0$. The row is filled in by adding more error patterns $\mathbf{e}_j$ with ever-increasing weight (Hamming distance from $\mathbf{0}$): all N single-error patterns, all $N(N-1)/2$ double error patterns, etc., up to a total of 2^{N-K} patterns. The generic entry $\mathbf{a}_{i,j}$ of the array is now found as the sum of the i-th first-column and the j-th first-row terms: $\mathbf{a}_{i,j} = \mathbf{a}_i + \mathbf{e}_j$. Eventually, the general appearance of $\mathbf{A}$ is as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0 = \mathbf{e}_0 & \mathbf{e}_1 & \cdots & \mathbf{e}_{2^{(N-K)}-1} \\ \mathbf{a}_1 & \mathbf{a}_1 + \mathbf{e}_1 & \dots & \cdots \mathbf{a}_1 + \mathbf{e}_{2^{(N-K)}-1} \\ \vdots & & \ddots & \vdots \\ \mathbf{a}_{2^K-1} & \mathbf{a}_{2^K-1} + \mathbf{e}_1 & \cdots & \mathbf{a}_{2^K-1} + \mathbf{e}_{2^{(N-K)}-1} \end{bmatrix} \tag{5.32}$$

This $2^K \times 2^{N-K}$ matrix, called the *standard array* of the code represents the most expedient way of "ordering" the channel output blocks so as to ease decoding. It is clear in fact that each column of the matrix (that is called a *coset*) is characterized by the *same syndrome*, since it has the same error pattern. The error pattern (the first element in each column) is called the *coset leader*, and it is this leader that we need to actually perform decoding. We do *not* need to store in the decoder the (huge) standard array. Rather, we have to build a table of all of the 2^{N-K} different syndromes, associated to their coset leaders. The decoder computes the syndrome, then looks up into the table to locate the corresponding leader. The leader represents the *minimum-weight* error pattern that is to be subtracted (that is, modulo-2 added) from the received block to obtain the most likely decoded codeword (and so the most likely source block).

Example 5.30

For the Hamming (7,4) code, the list of coset leaders (error patterns) has $2^3 = 8$ elements. We can build it considering 7-element binary vectors of increasing Hamming weight, starting from the all-zero vector. So the leaders are:

$\mathbf{e}_0$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$	$\mathbf{e}_5$	$\mathbf{e}_6$	$\mathbf{e}_7$
0000000	0000001	0000010	0000100	0001000	0010000	0100000	1000000

On the other hand, the codebook is

$$\begin{aligned}
\mathbf{a}_0 &= 0000000 \\
\mathbf{a}_1 &= 0001011 \\
\mathbf{a}_2 &= 0010101 \\
\mathbf{a}_3 &= 0011110 \\
\mathbf{a}_4 &= 0100110 \\
\mathbf{a}_5 &= 0101101 \\
\mathbf{a}_6 &= 0110011 \\
\mathbf{a}_7 &= 0111000 \\
\mathbf{a}_8 &= 1000111 \\
\mathbf{a}_9 &= 1001100 \\
\mathbf{a}_{10} &= 1010010 \\
\mathbf{a}_{11} &= 1011001 \\
\mathbf{a}_{12} &= 1100001 \\
\mathbf{a}_{13} &= 1101010 \\
\mathbf{a}_{14} &= 1110100 \\
\mathbf{a}_{15} &= 1111111
\end{aligned}$$

so that we would be in a position to build the standard array of the code. A task that we skip with no regret, since the only information that is actually needed to perform decoding is the list of all syndromes associated to the coset leaders as above. Using $\mathbf{s} = \mathbf{e}\mathbf{H}^T$, we get

$\mathbf{e}_0$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_4$	$\mathbf{e}_5$	$\mathbf{e}_6$	$\mathbf{e}_7$
0000000	0000001	0000010	0000100	0001000	0010000	0100000	1000000
000	001	010	101	011	101	110	111
$\mathbf{s}_0$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_6$	$\mathbf{s}_7$

Considering the distance properties of the code, no wonder that the leaders, representing error vectors, contain all the one-error patterns that we can have on a 7-bit word. With the table above, we are ready to do decoding once we have computed the syndrome of the received hard-detected word $\mathbf{d}$.

AS another example of a popular block code, we mention the (23,12) code discovered by Marcel Golay in the 60s (with a rate r=12/23 that is very close to 1/2) with a minimum distance equal to 7. The parity-check matrix of the code is $\mathbf{H} = \left[\mathbf{P}^T | \mathbf{I}_{11}\right]$ where the parity

matrix is

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{5.33}$$

The Golay code shares a nice property with the (7,4) Hamming code: they are both *perfect* codes. A perfect code is a code wherein the codewords are maximally equispaced, i.e, the distance between any codeword and its nearest neighbor is always the same and is of course equal to the minimum (Hamming) distance of the code. Apart form trivial cases, this guarantees that the code has the maximum coding rate for a given minimum distance d_{min}, since it has the maximum number of codewords C for a given minimum distance constraint (since they are very well distributed in the 2^N-elements space of all possible words). This is formalized by the so-called *Hamming Bound*: for any code having $d_{min} \leq 2q + 1$, then

$$C \leq \frac{2^N}{\sum_{i=0}^{q} \binom{N}{i}} \tag{5.34}$$

where equality on both relations concerning the minimum distance d_{min} and the number of codewords C only holds for perfect codes. The trivial perfect code is... the repetition code.

5.1.2.5 Post-decoding BER We already know that the error-correcting capability of a code is driven by d_{min}. Assume that the decoder receives the hard-detected $\mathbf{d}$ coming form a BSC with raw symbol error probability p_e. We previously managed to derive the post-decoding BER P_E (4.53) for the repetition code. As a matter of fact, a similar relation holds for *any* code, provided that we interpret N of the repetition code in (4.53) as the d_{min} of a generic code. Unfortunately, the relation does not give us the exact BER, but only an *upper bound*:

$$P_e \leq \sum_{n=(d_{min}+1)/2}^{N} \binom{N}{n} p_e^n (1 - p_e)^{N-n} \tag{5.35}$$

Equality only holds for.. perfect codes, including repetition.

If on the contrary the codeword is sent over an AWGN with matched-filter detection as in Fig. 5.2, the situation s more complicated. This time, we can easily found a *lower* bound of the post-decoding BER P_E, that is relatively accurate for high signal-to-noise ratio:

$$P_e \geq \mathrm{Q}\left(\sqrt{\frac{2E_s}{N_0}} d_{min}\right) = \left(\sqrt{\frac{2E_b}{N_0}} d_{min} \cdot r\right) \tag{5.36}$$

When d_{min} is large, the intrinsic loss on E_s/N_0 (for the same value of E_b/N_0) caused by coding is more than compensated for by the code, so that the overall coding gain is (asymptotically) equal to $r \cdot d_{min}$.

5.2 Convolutional Codes and the Viterbi Decoder

Block codes are still widely used in many ICT applications. For example, Reed-Solomon codes (a particular class of codes for channels with erasure) are used in Hard-Disk drivers, in digital television, in CD / DVD players etc.. The second most important class of channel codes is the one represented by *convolutional encoders with Viterbi decoding* and its many variants (punctured codes, Ungerböck's codes and so on).

5.2.1 Nonsystematic Convolutional Encoders

A convolutional encoder is a digital circuit that computes the *convolution* of the incoming binary data stream $b[k]$ with what we may call a certain impulse response $h[k]$ to give the stream of the coded symbols $a[n]$. Many variants of encoders exist and are currently used in the practice. One of the simplest class is that of feedforward (non recursive) non-systematic encoders with rate $1/N$, a sample of which is represented in Fig. 5.5 for $N = 2$. It is apparent that the two binary signals $a_0[k]$ and $a_1[k]$ are produced by two linear FIR filters (see). The main difference wrt the general architecture in Fig. 2.19 is that here all products and additions are to be intended as modulo-2 operations, but apart from this we see that the relation that is implemented is

$$a_i[k] = b[k] \cdot h_i[0] + b[k-1] \cdot h_i[1] + ... + b[k-K] \cdot h_i[K] \;\; i = 0, 1 \qquad (5.37)$$

just like with any FIR filter. Indicating explicitly the binary operations, we have

$$a_i[k] = b[k] \otimes h_i[0] \oplus b[k-1] \otimes h_i[1] \oplus ... \oplus b[k-K] \otimes h_i[K] \;\; i = 0, 1 \qquad (5.38)$$

The two impulse responses $h_0[m]$ and $h_1[m]$ have length K that is called the *constraint length* of the code (in practice, the number of consecutive input symbol that determine the value of an output symbol). We can come to the simpler final structure of the convolutional encoder shown in Fig. 5.6 considering that i) the delay line can be shared by the two "binary filters"; ii) the values of the $h_i[m], m = 0, ..., K-1$ are binary: if $h_i[m]$ is 1, then the delayed-by-m input bit is directly used to compute the output of the encoder according to (5.38), and a corresponding connection is present in the scheme in Fig. 5.6. If the value is 0 no connection for the delayed bit appears in the scheme; iii) the encoded stream $a[n]$ is obtained as the simple interleaving of $a_0[k]$ and $a_1[k]$, so that *two* coded binary symbols are produced for each input information bit: $a[n] = a_{|n|_2}[n//2]$. The most concise way of identifying a rate-$1/N$ encoder is therefore specifying N binary strings that represent the values of $h_i[m]$, $i = 1, ..., N$. For the rate-1/2 encoder of Fig. 5.6, the two strings are 111 and 101, respectively. As a practical example, Fig. 5.7 shows the specification of the $K = 7$, $r = 1/2$ convolutional encoder for the digital European terrestrial television standard DVB-T. The configuration strings are 1011011 1111001 which, translated into base 8 for conciseness, are shown in the figure.

In addition to the feedforward encoder we have just described, a feedback structure is often encountered in the practice, especially when it comes to the *concatenation* of codes (see Sect. 5.3). Exactly as with filters, saying "feedback structure" is equivalent to saying

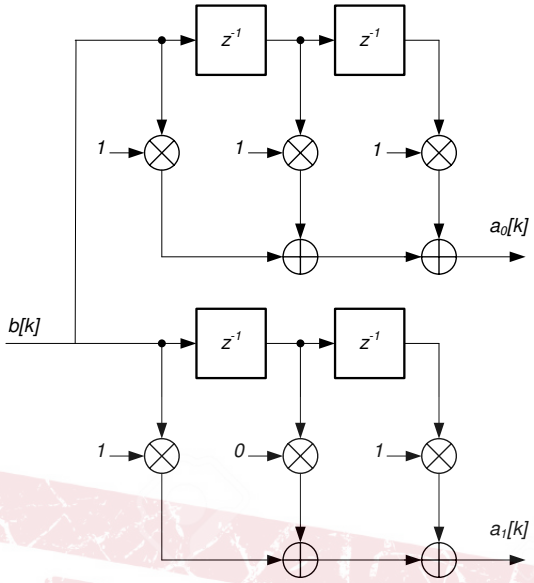


Figure 5.5 Generation of encoded symbols in a convolutional encoder with r=1/2 and K=3

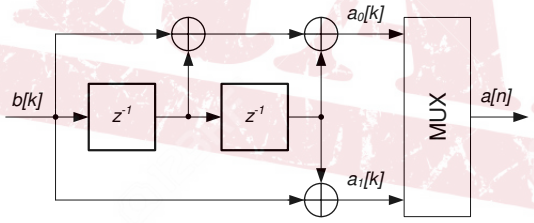


Figure 5.6 Implementation of a nonsystematic nonrecursive r=1/2, K=3 convolutional encoder

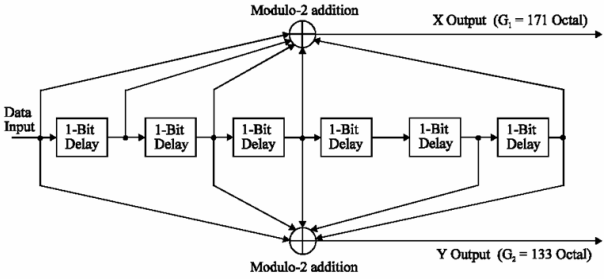


Figure 5.7 Convolutional encoder of the DVB standards for digital television

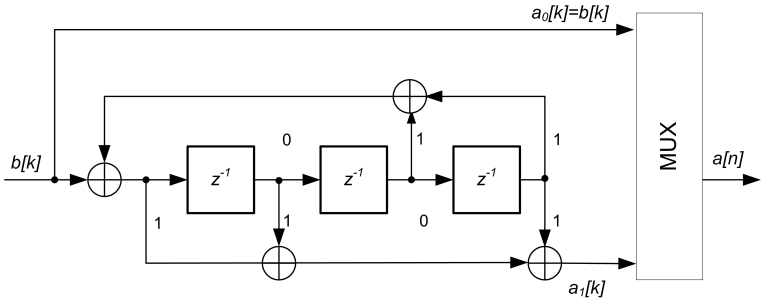


Figure 5.8 3gpp Systematic Recursive r=1/2, K=3 convolutional encoder

recursive input-output relation. Fig. 5.8 shows an example of a *systematic recursive* encoder with rate 1/2 and constraint length $K = 4$. The encoder is systematic because the even-index output of the encoder is $a[2k] = a_0[k] = b[k]$. Using transfer-function based techniques, we can also show that the odd-numbered output $a[2k + 1] = a_1[k]$ is produced by the encoder with the following recursive equation:

$$a_1[k] = a_1[k-2] \oplus a_1[k-3] \oplus b[k] \oplus b[k-1] \oplus b[k-3] \tag{5.39}$$

The encoder is now IIR with *infinite-length* impulse response, and the convolution is implemented through the recursive equation (5.39) and not directly with the impulse response as in (5.38). The encoder is specified through the coefficients of the feedforward part and of the feedback part separately. For the circuit in Fig. 5.8, the binary pattern describing the feedforward connections is 1101 while the feedback part is (1)011. In the feedback connections specification, the first digit (that represents the un-delayed output of the initial adder) is always equal to 1 (as is in the denominator of normalized transfer functions of linear recursive digital filters).

In general, codes generated by feedback encoders can be made equivalent to codes generated by feedforward ones, in the same sense already discussed for block codes.

More complicated structures of encoders (be them feedforward or feedback) are those for rates of the kind M/N with $M \neq 1$. But the main concepts are the same as those that we will develop for the simpler rate-$1/N$ encoder, so that we will not take them into consideration.

At the end of the section, we remark a fundamental difference between block and convolutional codes: in the former, the information bits are segmented in blocks of length K, and all the transmitter encoding and receiver de-coding processing takes place within the finite time horizon of a *block*. The convolutional encoder (5.38) or (5.39), on the contrary, does not have any intrinsic limitation to the length of the input and/or to the output bit sequence, which can be of an arbitrary length N, even very large - we can call them *sequence* encoders. This difference will reflect into the decoding algorithm that we introduce in the next section.

5.2.2 Decoding a Convolutional Code

We have thus slowly come to the two fundamental issues: i) given a convolutional encoder, how can I decode back the information bit stream with maximum protection against channel noise? ii) given a convolutional encoder and its (optimum) decoder as in i), how can we compute or evaluate its BER performance (and so, given a certain decoding complexity

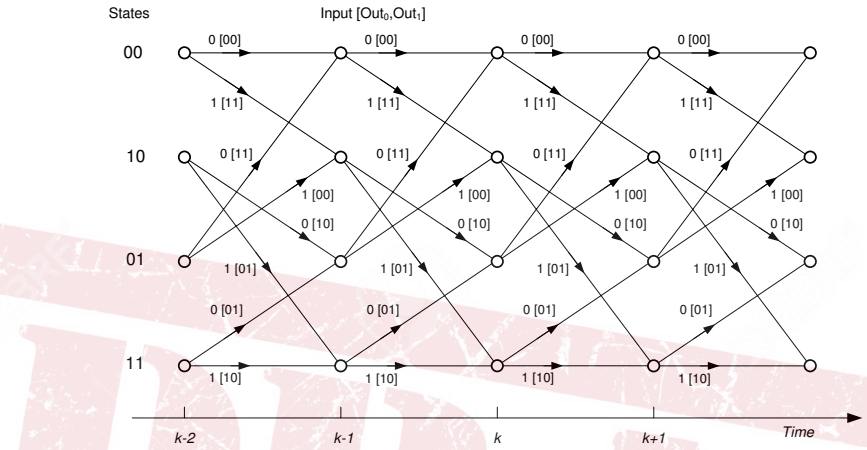
I can afford, what is the best code)? Concerning question #1, the funny thing is that convolutional encoders were invented by Elias in the 50's *before* the optimum decoder that is universally used nowadays was actually introduced. We allude to the celebrated Viterbi Decoder (VD) to perform efficient decoding of a convolutional code. Generally speaking, optimum decoding of a convolutional code can be carried out following the same MAP or equivalent MV approach that led us to the distance minimization criterion (5.8). The main problem here is that, contrarily to block codes, the "codeword" of a convolutional encoder has theoretically an unbounded length - in practice it very very large, as large as a whole frame of data. Exhaustive minimum distance decoding is therefore out of the question. We may say that MV decoding of a convolutional code amounts to maximum-likelihood sequence estimation (MLSE): the decoder has to find that (arbitrary-length) sequence of channel symbols $\hat{a}[n]$ (that we can relate to a specific sequence of source bits) amidst those allowed by the code, whose (Hamming or Euclidean) distance is the minimum with respect to the sequence of (hard or soft) observed symbols. This is actually what the VD does.

To understand how the VD works, we introduce the so-called *trellis representation* of the time evolution of a convolutional encoder. We will use in our discussion the encoder in Fig. 5.5 as a paradigm for all necessary developments and remarks. We see that the output of the encoder is determined as long as i) the current value of the input information bit $b[k]$ is known, and ii) all of the previous values of input bits $b[k-1], ..., b[k-K-1]$ in the shift register of the encoder are known. The string of previous bits is the *state* of the encoder that can be regarded as a *finite-state* sequential digital circuit (Mealy machine). The number of states N_s of the machine is of course the number of different $(K-1)$-bit strings that can be stored in the shift register, i.e., $N_s = 2^{K-1}$. Every time a new bit is input to the machine, the state changes according to the simple right-shift function, and a new output is computed based on the value of the input and of the value of the state.

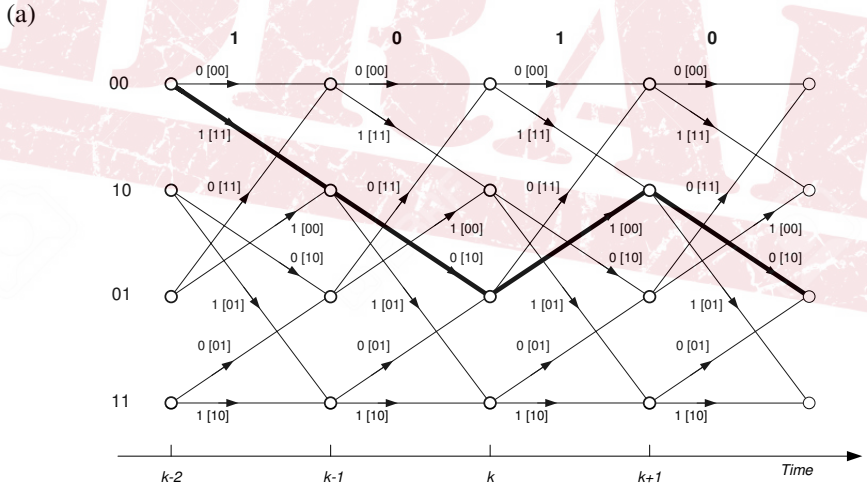
The possible evolution of the encoder is thus represented in a diagram like the one in Fig. 5.9 (a) for our customary sample encoder. On the horizontal axis we represented time, indexed by k as usual, whilst on the vertical axis we put the possible $N_s = 4$ states of the encoder represented as small circles and labeled by the contents of the shift register at that time. The branches that join some states at consecutive time instant represent *state transitions* as caused by the shift function implemented by the register. Of course, not all transitions are possible, but only those determined by the specific structure of the encoder (in particular, those determined by the right-shift function). The labels on the branches have a double meaning: the first digits represents the value of the input $b[k]$ that generated that transition, and the values between braces are the values of the outputs $a_0[k], a_1[k]$ produced by the encoder starting from that particular state at time $k-1$ and with that particular input.

As is shown in Fig. 5.9 (b), the specific time evolution of the encoder subject to a particular input string (here 1010) is represented by a specific *path* taken amidst all of the possible branches in the trellis; we assume here that the path has come to time $k-2$ in state 00.

Assume now that our coded logical binary symbols $a[n] \in \{0, 1\}$ are re-mapped to antipodal channel symbols $\alpha[n] \in \{-1, 1\}$ and sent out onto an AWGN channel. The decoder for our convolutional code has a well defined task: upon reception of a sequence of $2N$ noisy versions $r[n] = \alpha[n] + w[n]$ of the coded bits $\alpha[n], n = 0, 1, ..., 2N-1$, with $w[n]$ AWGN, find that particular path in the trellis (hence that particular sequence of N information bits $b[0], b[1], ..., b[N-1]$) that with highest probability has generated what has been actually observed. If we assume equiprobable and independent information bits $b[k]$ and an AWGN channel, the solution of this estimation problem, called *Maximum Likelihood Sequence Estimation*, is just the Viterbi Detector. We do not need here to re-derive the VD



(a)



(b)

Figure 5.9 Trellis description of the $r = 1/2$ convolutional encoder in Fig. 5.5 (a); Specific path in the trellis caused by the specific input string 1010 (b)

that is well documented on each textbook of digital communications, nor we try to give a didactic explanation of it. We just intend to revise its functioning, and in particular review the basic notions of *survivors*, *branch metrics*, *Add-Compare-Select*.

As is known, the VD (or Viterbi Algorithm, VA) has a time-recursive formulation that makes it low-complexity with respect to an exhaustive search on all of the possible 2^N input bit sequences. Assume that the decoder has come somehow to time $k-1$ in its reconstruction of the path in the trellis. If it is not the final time $N-1$, then the decision on the most likely path has not come to an end yet. As is depicted in Fig. 5.10 (a), the VD at time $k-1$ has retained N_s "candidate" paths (one for each state) identified by the candidate information sequences $\hat{\mathbf{b}}_i[k-1] = \left[\hat{b}_i^{(k-1)}[0], \hat{b}_i^{(k-1)}[1], ..., \hat{b}_i^{(k-1)}[k-1]\right]$, $i = 0, 1, ..., N_s - 1$, that are called *survivors* (we'll see in a while the reason for this name). At time $k-1$, each survivor also owns a measure, or more precisely a *metrics* $\delta_i[k-1], i = 0, ..., N_s - 1$ it has accumulated up to the present time about its (smaller or larger) *likelihood* of being the final estimation. Such metrics are reported in Fig.5.10 (a) on the right-hand side of the trellis for each survivor.

Starting form this configuration, what the VD does is the *extension* of such survivors up to time k. This can be done as soon as the two noisy symbol replicas $r[2k-2] = \alpha_0[k-1] + w[2k-2]$ and $r[2k-1] = \alpha_1[k-1] + w[2k-1]$ are received (the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$ for $a[n]$ to $\alpha[n]$is assumed). This allows the VD to compute first the so-called *branch metrics* on the trellis transitions as follows:

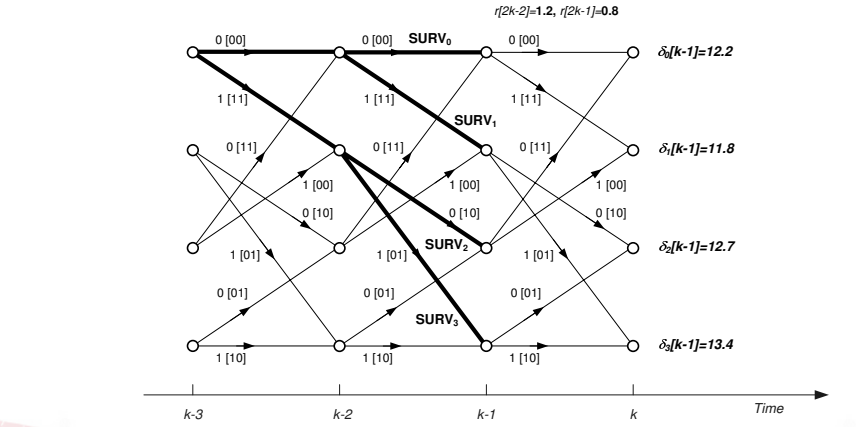
$$\lambda^{(i,j)}[k] = \left(r[2k-2] - \hat{\alpha}_0^{(i,j)}[k-1]\right)^2 + \left(r[2k-1] - \hat{\alpha}_1^{(i,j)}[k-1]\right)^2 \qquad (5.40)$$

In this equation, $\hat{\alpha}_0^{(i,j)}[k-1]$ and $\hat{\alpha}_1^{(i,j)}[k-1]$ are the values of the (re-mapped) coded symbols that are produced by the encoder when it is in the state i at time $k-1$ and goes into state j at time k - they are specific of the (i,j) transition. Figure 5.10 shows on top, for each bit interval, the received noisy values of $r[2k-2]$ and $r[2k-1]$ that are used by the VD to compute the branch metrics that we showed on each transition in the trellis in (b) from time $k-1$ to time k.

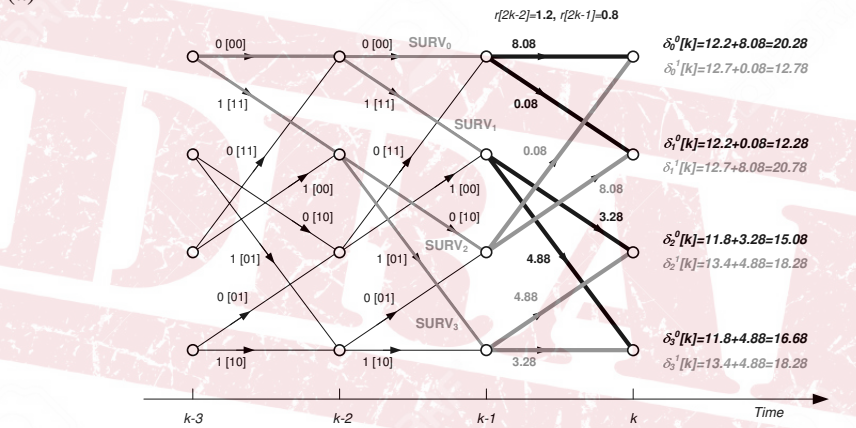
The branch metrics allow to extend the survivors and update the accumulated metrics. If we focus in fact on the generic arrival state j at time k, we see that two branches end up into such a state coming from two departure states i_0 and i_1 at time $k-1$. For instance, we see that the branches that end into state 00 come from states 00 and 01 at the preious time step. We can build two "candidate" sequences poised to become the new survivor of state j at time k by simply extending along that branches the two survivors of states i_0 and i_1 at time $k-1$, as is indicated in Fig. 5.10 (b), where the two "converging" branches in each arrival state are shown in black and in gray. The VD *adds* the two branch metrics computed at time k to the previous accumulated metrics of the two survivors that at time $k-1$ were in the states i_0 and i_1 to give two accumulated metrics of two *candidate* survivors, both ending in state j at time k:

$$\begin{aligned} \delta_j^0[k] &= \delta_{i_0}[k-1] + \lambda^{(i_0,j)}[k] \\ &\qquad . \\ \delta_j^1[k] &= \delta_{i_1}[k-1] + \lambda^{(i_1,j)}[k] \end{aligned} \qquad (5.41)$$

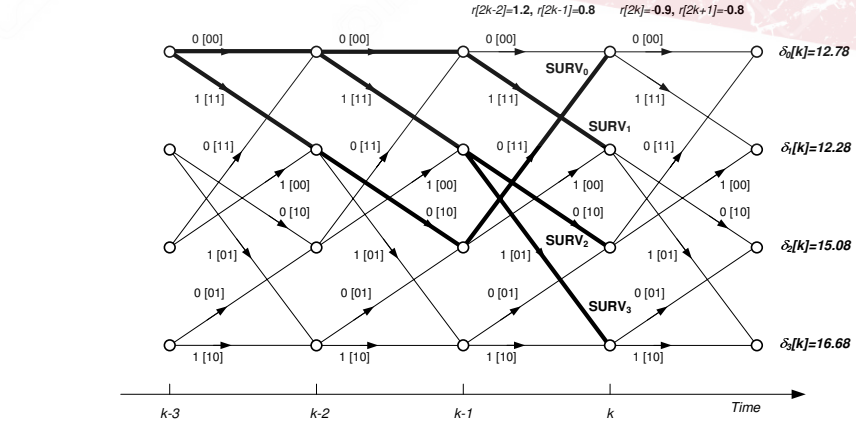
Figure 5.10 (b) shows the computed branch metrics on each transition leading to the diverse states at time k, as well as the two resulting candidate accumulated metrics $\delta_j^0[k]$ and $\delta_j^1[k]$ on the right-hand side, for each state. The two colors refer of course to the two candidate survivors bearing the same transition colors, respectively.



(a)



(b)



(c)

Figure 5.10 Survivors at time $k-1$ on the trellis (a); ACS process (b); Survivors at time k (c)

Then, for each state, the VD *compares* the two candidate accumulated metrics $\delta_j^0[k]$ and $\delta_j^1[k]$ to find the minimum one and eventually *selects* the candidate path that has the minimum accumulated metrics (i.e., that has the minimum *distance* from the observed noisy signal samples), $delta_j[k] = \arg\min_i \delta_j^i[k]$. This sequence (path) *survives* the comparison after the ACS procedure just describes, the other is discarded *for ever* since *any* future candidate path in the search of the VD that will possibly go out of the state j at time k will always have a smaller metric if obtained as an extension of the survivor than of the "perished". It is just this *select* operation that prevents the complexity of the VA from growing exponentially with time (as does on the contrary the exhaustive distance evaluation on all of the possible sequences produced by the encoder). For na information sequence of total length N, the complexity is therefore equal to $N\ cdot N_s$ rather than 2^N!

At the end of this ACS (add-compare-select) process, the VD has new survivors $\hat{\mathbf{b}}_i[k] = \left[\hat{b}_i^k[0], \hat{b}_i^k[1], ..., \hat{b}_i^k[k]\right], i = 0, 1, ..., N_s - 1$, as shown in Fig. 5.10 (c). Note that the $i - th$ survivor at time k is not necessarily the extension of the i-th survivor at times $k - 1$[3]. On the contrary, in the extension procedure the survivors are often "scrambled" in the sense that what used to be the survivor for a certain state may very well become the first part of the survivor of another state after the extension is done. This is seen in Fig. 5.10 (c) that shows the status of the decoder at time k after all ACS's for all states are done.

When $k = N$, a final selection between the survivors is made, and the "survivor of the survivors" that has the minimum accumulated metrics $\delta_j[N-1]$ is finally selected as the most likely sequence of information bits. Of course, when N is very large, storing all survivors from the beginning of decoding till the very end may be unfeasible. Fortunately, if we trace the survivors from time k backwards, we see that they tend to "bunch" or *merge* back into a unique survivor all of them stem out of (Fig. 5.11). When this happens (and experience suggest that it happens with high probability whenever we go back into the trellis for $4 \div 5 \cdot K$ information bits, K constraint length of the encoder) all decisions on the information bits belonging to the unique section of the survivor can be safely taken in advance since, whatever the final survivor of the survivors is, it will always stem out of that section and it will always lead to the same final decisions. The decoder operates almost in real time, leading only to a decoding delay (that is also called *decision depth*) equal to 4 to 5 times the constraint length K due to the trace-back above.

5.2.3 BER Performance

We are now ready to tackle the second fundamental question of this subsection, that is, how can we compute or evaluate the BER performance of the VD for a convolutional code with, say, AWGN ? The issue has not a simple close-form solution, so that often direct measurements or computer simulation is used to give an answer to our question. But a simple bound of the BER exist, and it is worth discussing here. The bound is also useful in the design of "good" convolutional codes as we will see in a while.

We have seen that the VD tries to reconstruct the path that the encoder has taken in the trellis when driven by a particular string of input infirmation bits. If we reconsider the description of the VD above, in particular of the ACS process and how the accumulated metrics builds up, we easily recognize that when the noise component is small (that is, when $r[2k] \simeq \alpha_0[k]$ and $r[2k+1] \simeq \alpha_1[k]$), the survivor that corresponds to the actual sequence

[3]This is also the reason why we need a second (time) index in the elements of the survivors in addition to the state identifier

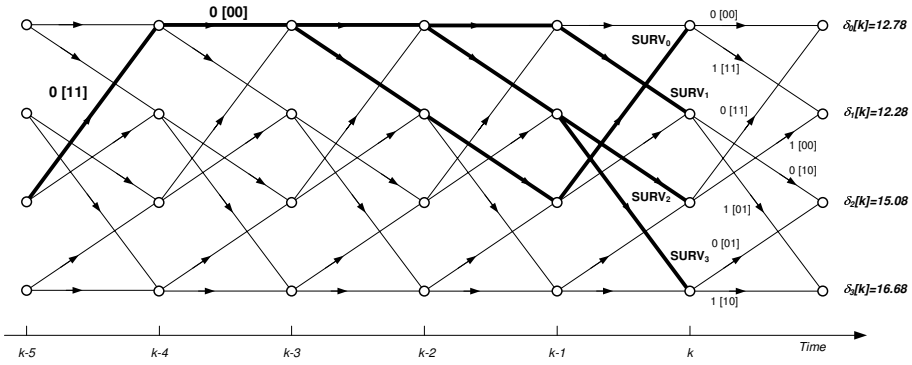


Figure 5.11 Merging of survivors in the Viterbi Detector

of information bits accumulates a metrics that is practically zero, whilst the metrics of the others grow unboundedly since the sequence of tentative coded symbols $a_0^{(i,j)}[k]$ and $a_1^{(i,j)}[k]$ in (5.40), apart from sporadic cases, do not correspond to the actual sequence as it does on the "survivor of the survivors". Some reflections on the particular expression of the branch metrics (5.40) suggest us that it has the meaning of a *distance*, actually a *squared Euclidean distance* between the received sample and the pair of tentative coded symbols $a_i^{(i,j)}[k], i = 0, 1$. The accumulated metrics (5.41) has thus the meaning of squared Euclidean distance between the *sequence* of received signal samples $r[0], ..., r[2N-1]$ and each survivor. The final "survivor of the survivors" is therefore that sequence that has the minimum overall distance from the received signal samples. When does the VD make a decoding error? In the light of our previous discussion, we can say that an *error event* is produced whenever, due to a large noise component, a path in the trellis exist (hence an information bit sequence exist) that is closer to the received signal than the correct path is. This is the case for the grey path in Fig. 5.12. The VD momentarily takes a wrong route to hopefully re-merge into the correct path after an unpredictable time that depends on the strength of the noise and on the property of the code (see Fig. 5.12 again). During this error event, the decoder produces a burst of errors whose length is random. The more frequent are error events, the larger the BER of the decoder will be.

What is the main property of the code that determines such frequency ? It is quite easy to see that error events are infrequent if all possible coded symbols sequences corresponding to paths in the trellis are sufficiently "separated", like with any code (see the discussion about the design criteria of algebraic codes in the previous section). Separated means "with high mutual Euclidean distance", so that selecting a wrong path in the trellis is not so frequent even in the presence of strong noise. And here we have come to the main performance parameter of the code: the minimum squared distance that separates distinct paths in the trellis. If this distance is small, it is relatively easy for the VD to go along a wrong path - error events are frequent and the BER performance is bad. Finding a good code amounts therefore to finding a trellis whose *minimum squared Euclidean distance* is high enough. In the case of binary signaling on the AWGN, this is equivalent to finding codes with good Hamming distance. In particular, the minimum Hamming distance for (arbitrarily) long code sequence pairs on a trellis is usually denoted with d_{free}. From the previous discussion, we understand that the BER performance of the VD with soft-input decoding on the AWGN channel is basically dictated, for high values of the E_b/N_0 ratio, by d_{free}. Without entering

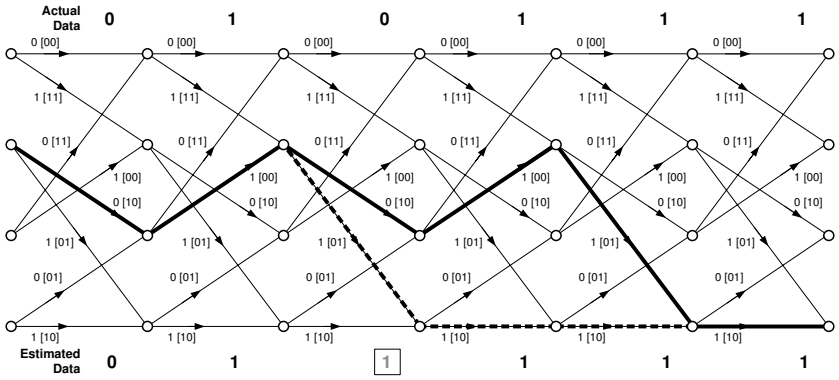


Figure 5.12 An error event in the Viterbi Detector causing 1 source error

K	$(h_0[k])_8$	$(h_1[k])_8$	d_{free}
3	5	7	5
4	15	17	6
5	23	35	7
6	53	75	8
7	133	171	10
8	247	371	10
9	561	753	12

Table 5.1 Table of best rate-1/2 convolutional codes

into details, it is easy to see that

$$P_e \simeq N_{avg} \cdot Q\left(\sqrt{r \cdot d_{free} \frac{2E_b}{N_0}}\right) \tag{5.42}$$

where N_{avg} represents the average number of wrong bits in an error event (that is usually a small number larger than 1). Neglecting such parameter, it is easy to see that the coding gain of the convolutional code is equal to $20\log(r \cdot d_{free})$ dB. A low coding rate has to be compensated by a more than proportional increase in the d_{free} to yield a net increase in the coding gain. Tables of optimum codes with assigned rate and constraint length are easily available in the literature. They were found after exhaustive computer search on all of the possible $h_i[k]$ to be used in (5.38). A sample is given in Tab. 5.1 for rate-1/2 codes. The set of values of $h_i[k]$ (the so called *generators*) is given, together with the value of the minimum Euclidean distance d_{free} between any two (long) sequences in the codebook, and with the value of N_{avg}. To avoid long binary strings, the sequence of the values of $h_i[k]$ is expressed as a base-8 number: each octal digit is to be interpreted as the corresponding sequence of 3 binary values.

To sum up with all of the concepts that we have introduced in this subsection, Fig. 5.13 shows the BER performance of the optimal codes in Tab. 5.1, including the one in Fig. 5.7. The performance improvement for increasing d_{free} (as well as increasing complexity N_s) is apparent.

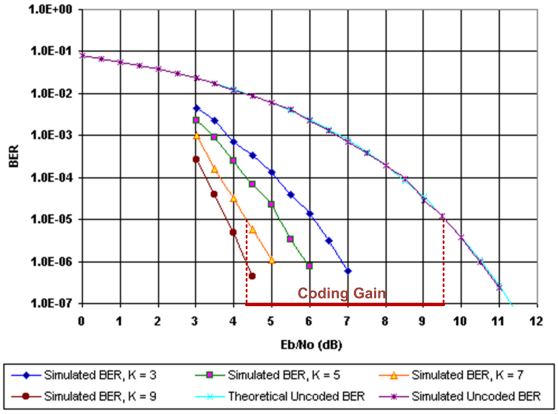


Figure 5.13 Simulation results for the BER of convolutional codes on the AWGN channel

5.2.4 Probabilistic (Soft-Output) Decoding: The BCJR algorithm for Convolutional Codes

The Viterbi algorithm is not the only way to decode a convolutional code. In particular, the output of the Viterbi Decoder (VD) is directly an estimate of the binary information bitstream $b[k]$, as we have seen in the previous subsection. We can say that the VD is soft-input, binary-output (SIBO). Other decoding algorithms were developed in the past that, in addition to the hard output also provide a continuous-valued (*soft*) output associated to each decoded bit: soft-input, soft output (SISO).

One such decoder is the well-know BCJR algorithm (from the names of the inventors Bahl, Cocke, Jelinek, Raviv) to decode *finite-size* blocks of symbols produced by a convolutional encoder (not a *sequence* estimator). Before saying what the continuous value associated to the (estimate of) $b[k]$ is, let us introduce some notation. With the usual notation, we consider a block of K information bits $\mathbf{b} = [b[0], b[1], ..., b[K-1]]$ that is convolutionally encoded by a rate-r code into a block of N (with $N = K \cdot r$ an integer) coded binary symbols $\mathbf{a} = [a[0], a[1], ..., a[N-1]]$. The N coded symbols $a[n]$ are mapped onto antipodal, ± 1 symbols $a_{\pm} n$ and are sent onto an AWGN channel with binary signaling and matched filter detection. Assuming no ISI, the outputs of the matched filter corresponding to the coded symbols are $r[n] = a[n]_{\pm} + w[n], n = 0, 1, ..., N-1$, with $w[n]$ discrete-time white Gaussian noise with variance $\sigma^2 = (2E_s/N_0)^{-1}$. The received signal block is thus, with self-evident notation, $\mathbf{r} = \mathbf{a}_{\pm} + \mathbf{w}$. Due to the memory of the encoder, the different values of $\mathbf{r}$ are *not* independent from each other. The structure of the convolutional encoder easily tells us that $a[n]$ depends on the history of the encoder, i.e., $a[n-1], a[n-2], \ldots$. From this, we see that $a[n-1]$ depends in its turn on $a[n]$, i.e., something that belongs to its own "future", so that in general it makes sense to take into consideration any possible information regarding the generic bit $b[k]$ coming from *all* of the observed signal samples in the block $r[n]$, be them in the "past" or in the "future" with respect to time k - something that the VD does not do since it operates in real time and does not have access to the future of $b[k]$.

This remark leads us to the introduction of *block-wise maximum a-posteriori probability* of $b[k]$ according to the following criterion:

$$\hat{b}[k] = \arg\max_{x} \left(\Pr\left\{ b[k] = x | \mathbf{r} \right\} \right) \tag{5.43}$$

The BCJR algorithm does this, and in addition gives us also an indication of the *reliability* about each decision. The soft-output of the decoder is in fact the quantity

$$L(b[k]) \stackrel{\triangle}{=} \ln \frac{\Pr\left\{ b[k] = 1 | \mathbf{r} \right\}}{\Pr\left\{ b[k] = 0 | \mathbf{r} \right\}} \tag{5.44}$$

The reason why we regard this as a reliability metrics is easy to understand. Assume that $b[k] = 1$. *Before* observing $\mathbf{r}$, the two *prior* probabilities of $b[k]$ being equal to 0 or 1 are equal - no decision can be made. *After* the observation of $\mathbf{r}$, the two probabilities change, they become the *posterior* probabilities, and in the general they are (much) different from each other. Therefore, a decision about the value of $b[k]$ being pretty safe (reliable), means that upon observation of $\mathbf{r}$, the $\Pr\left\{ b[k] = 1 | \mathbf{r} \right\}$ at the numerator is much larger than the $\Pr\left\{ b[k] = 0 | \mathbf{r} \right\}$ at the denominator. The ratio is much larger than 1 and the $\ln$ is large and positive. If $b[k]$ is equal to 0, by the same reasoning we see that $L(b[k])$ is large and negative: in both cases the so called *log-a-posteriori-probability-ratio* (LAPPR) $L(b[k])$ has a large amplitude (absolute value), and this indicates high *reliability*. Having a LAPPR close to 0 on the contrary means that the two probabilities are not so different, and so the decision about $b[k]$ is not completely safe. The *sign* of $L(b[k])$ is in practice the final (hard) decision on the information bit: if $L(b[k]) > 0$, then $\hat{b}[k] = 1$, and the converse if $L(b[k]) < 0$.

Example 5.31

The notion of LAPPR does not necessarily need to be related to an encoded signal. Assume that we have the usual binary antipodal uncoded signaling on AWGN, so that the received signal at the output of the matched filter is $r[n] = a_{\pm}[n] + w[n]$ with $a_{\pm}[n]$ directly representing the information source bits, $a_{\pm}[n] = 2b[n] - 1$. There's no statistical dependence between $r[n]$ and $r[m]$, $n \neq m$, so that $L(b[n])$ only depends on $r[n]$ and is trivial to compute. To do so, we only need to observe that

$$L(b[n]) = \ln \frac{\Pr\left\{ b[n] = 1 | \mathbf{r} \right\}}{\Pr\left\{ b[n] = 0 | \mathbf{r} \right\}} = \ln \frac{\Pr\left\{ b[n] = 1 | r[n] \right\}}{\Pr\left\{ b[n] = 0 | r[n] \right\}}$$

$$\ln \left(\frac{f_R(r[n]|b[n]=1) \Pr\{b[n]=1\}}{f_R(r[n])} \frac{f_R(r[n])}{f_R(r[n]|b[n]=0) \Pr\{b[n]=0\}} \right)$$

$$= \ln \frac{f_R(r[n]|b[n]=1)}{f_R(r[n]|b[n]=0)} \tag{5.45}$$

where we have used Bayes' rule, we assumed that the two levels of the source bits are equiprobable, and where $f_{\mathbf{R}}(\cdot)$ indicates a pdf of the received signal $\mathbf{r}$. Now, the two conditional pdf's that appear in (5.45) are both *Gaussian* with the same variance $\sigma^2 = (2E_s/N_0)^{-1}$, and with a mean value that is given by $a_{\pm}[n] = 2b[n] - 1$. Using the familiar expression of the Gaussian pdf and computing the logarithm, we get

$$L(b[n]) = -\frac{(r[n]-1)^2}{2\sigma^2} + \frac{(r[n]+1)^2}{2\sigma^2} = \frac{2r[n]}{\sigma^2} = \frac{4E_s}{N_0} r[n] \tag{5.46}$$

This also shows that the optimum MAP decision rule on $b[n]$ is a hard-decision on $L(b[n])$, that is, just a hard decision on $r[n]$ as we already knew.

What we still have to understand is *how* the BCJR decoder derives the LAPPRs. We leave such details to Appendix A, and we just remark here that teh value of $L(bk)$ for each bit is derived after *two* algorithms that "scan" the code trellis much like what a VD does. The first algorithm runs over the received signal block $\mathbf{r}$ from the beginning to the time corresponding to bit $k-1$ as an ordinary VD to keep into consideration the dependence of $b[k]$ on *past* channel outputs; the second one runs on the contrary *backward in time* from the end to the beginning to account for the possible dependence of $b[k]$ on *future* channel outputs. The step-by-step "accumulated metrics" of the survivors of the two VD-like algorithms, as well as a further value accounting for the "branch metrics" at the time instant k, are combined together to finally give the numerator and the denominator of $L(b[k])$ we seek for and, based upon that, the estimate of $b[k]$, $\hat{b}[k]$. According to this description, we can evaluate the complexity of the BCJR as roughly double that of the VD for the same code.

Does the BCJR decoder yield the same performance as a VD? Very similar indeed, if the encoded bits are independent and equiprobable, since in that case MAP decoding of a single bit or MLSE decoding of the whole finite-length sequence are equivalent. There might be slight differences due to the implementation (decoding-depth, finite arithmetics, and so on). In general, there is no real advantage of going for soft-output decoding, unless the soft output reveals expedient to do something better than simple one-shot hard detection. This is indeed the case for the celebrated *turbo* algorithm for iterative detection of concatenated convolutional codes, as we show in the next subsection.

5.3 Iterative (Turbo) Decoding of Concatenated Convolutional Codes

The real breakthrough in modern coding techniques happened in 1993 at a conference in Geneva, Switzerland, when two French researchers unveiled for the first time their new channel codes with remarkably good performance. Until then, the best known channel codes for intermediate-to-low coding rates in practical use was the *concatenation* of an outer block code of rate r_o with algebraic decoding (Reed-Solomon code), cascaded with an inner convolutional code of rate r_i with Viterbi decoding (Fig. 5.14 (a)). An example of this technology is the historical concatenated code of digital TV DVB with a (204,188) RS code with $r_o = 188/204$ and an inner convolutional code of variable rate obtained by suited modification of the well-known $K = 7$, $r_i = 1/2$ encoder in Fig. 5.7. The outer code is *low-redundancy* and its purpose is to reduce the long bursts of errors produced by an error event in the inner-code VD. By this arrangement, the overall rate is not changed too much by the outer code and is equal to $r = r_i \cdot r_o = 94/204$ when $r_i = 1/2$, and the usage of consolidated technologies leads anyway to a good composite (concatenated) code. The coding gain at 10^{-5} for the lowest-rate code with $r = 94/204$ is a good 7 dB (see Fig. 5.14 (b)), nonetheless the point representative of QPSK with this code that we show with label DVB-S in Fig. 5.38 is not so close to the Shannon limit.

5.3.1 The Turbo PCCC Encoder

The idea of somewhat *concatenating* two codes was already around. But the decoder of concatenated codes was definitely sub-optimal: it was just the cascade of the two decoders of the two codes as considered separately. In a sense, the decoder did not take much advantage of the *concatenation*. But Berrou and Glavieux added a fundamental ingredient to the recipe: *iterative decoding* with soft-output detection of the constituent codes.

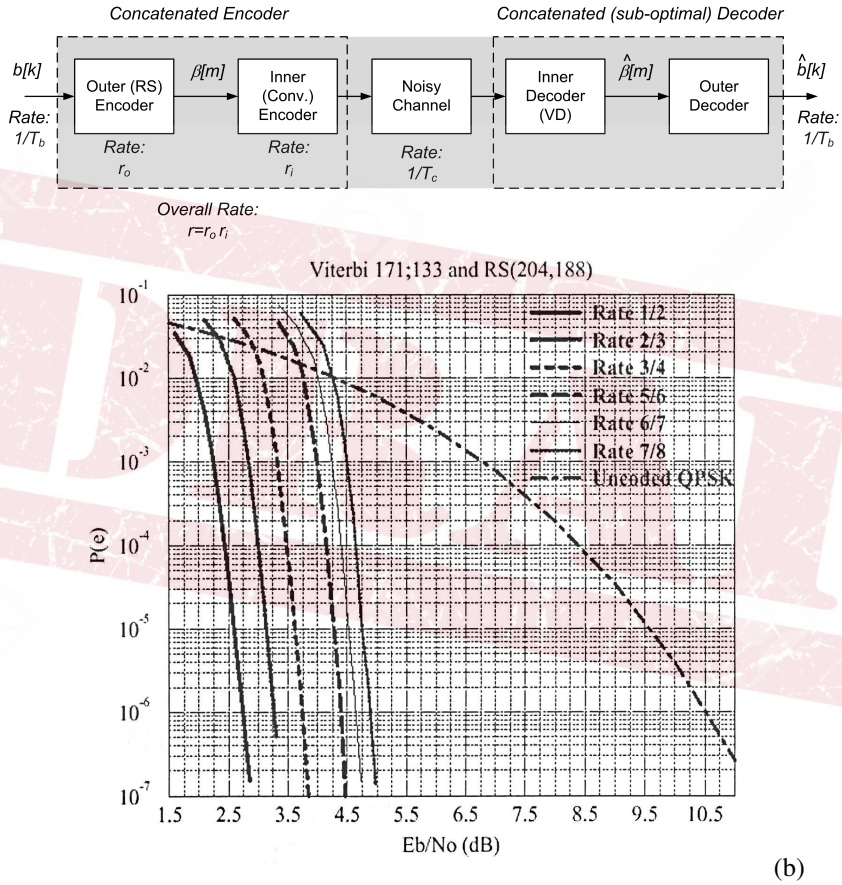


Figure 5.14 Schema (a) e curve di BER (b) del codice concatenato della televisione digitale DVB

Before entering into more details about the construction of a turbo code, we need re-working a little bit the BCJR algorithm, starting back from the definition of LAPPR:

$$L(b[k]) \triangleq \ln \frac{\Pr\{b[k]=1|\mathbf{r}\}}{\Pr\{b[k]=0|\mathbf{r}\}} = \ln \frac{f_{\mathbf{R}}(\mathbf{r}|b[k]=1)}{f_{\mathbf{R}}(\mathbf{r}|b[k]=0)} + \ln \frac{\Pr\{b[k]=1\}}{\Pr\{b[k]=0\}} \tag{5.47}$$

where we have used Bayes rule as in (5.45). The first term in (5.47) is a *Log-Likelihood Ratio* (LLR), whilst the second ratio should be 0 since it represents a term depending on the *a-priori* probabilities of the information bits. With good source compression, the probabilities of the two binary values are equal, and the log-ratio is 0 - this a-priori LAPPR component appears to be useless. But assume that we use an *additional decoder* that has side-information about the source bit, as for instance coming form another independent communication channel. In the light of such external (or *extrinsic*) information, we should no longer regard the two possible values of $b[k]$ as equiprobable, we should not consider the "a-priori" LAPPR component in (5.47) as vanishing, and we could possibly improve the realiabilty of bit decisions.

In reality, the side-information just comes in the form of an additional value of the LAPPR that we will call $L^e(b[k])$ where the superscript ^{e} means that this value has been derived with extrinsic means. We have to find a way to use this piece of *extrinsic information* as an additional input to our BCJR decoder to replace the useless a-priori value in (5.47) (adn we will see thios later on). First, we have to understand how we can get such piece of uncorrelated, extrinsic information, a sort of "second opinion" on the value of our information bit. The answer to the latter question is in Fig. 5.15 that depicts the simplest rate-1/3 *turbo encoder* made of the *parallel concatenation of convolutional codes* (PCCC turbo code). We see that two identical *constituent* recursive systematic convolutional (RSC) encoders with rate 1/2, just like the one in Fig. 5.8 are used. The encoder labeled RSC1 receives a block $\mathbf{b}$ of K information bits $b[0], b[1], ..., b[K-1]$ ordered in time as they are produced by the information source. The encoder RCS2 receives the block $\mathbf{b}_\pi$ that is produced by the *permuter* Π. $\mathbf{b}_\pi$ contains the same bits that appear in $\mathbf{b}$, but with a randomly-permuted order. If we call Π the permutation rule, than

$$\mathbf{b}_\pi = [b[\Pi(0)], b[\Pi(1)], ..., b[\Pi(K-1)]] \tag{5.48}$$

The permuter is actually the component that determines the information blocklenght, which is usually quite large - anywhere between a few hundreds to a few thousanda: we have devised a code with *large* blocks and using a design criterion that has an element of *randomness*, just like the random codes of the channel encoding theorem by Shannon!

The function of the permuter is to make the two blocks of bits appear as *uncorrelated*, so that the two *parity* bit blocks $\mathbf{p}$ and $\mathbf{p}_\pi$ produced by the two encoders are basically uncorrelated as well - they appear as having been produced by totally different information bits. Of course, the permutation law applied by the permuter Π (also called the *interleaver*) is not totally random. It follows a deterministic law that nonetheless gives the appearance of randomness: a pseudo-random interleaving law. The systematic output $\mathbf{b}$ of RSC1 is retained, and becomes the overall systematic output of the turbo encoder, whilst $\mathbf{b}_\pi$ from RSC2 is discarded. $\mathbf{b}$ is then bit-wise multiplexed (interleaved) with $\mathbf{p}$ and $\mathbf{p}_\pi$ to give the overall output of the systematic turbo encoder

$$\mathbf{a} = [b[0], p[0], p_\pi[0], b[1], p[1], p_\pi[1], ..., b[K-1], p[K-1], p_\pi[K-1]] \tag{5.49}$$

The size of the encoded symbol block is $N = 3K$ and the corresponding coding rate is 1/3. Figure 5.16 shows the rate-1/3 encoder standardized by 3gpp (www.3gpp.org) for use into 3G and 4G mobile cellular phones that exactly follows the construction we have just seen.

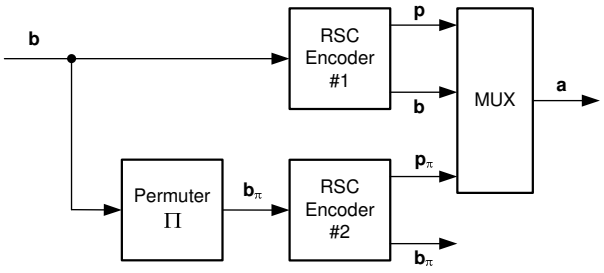


Figure 5.15 PCCC Turbo encoder

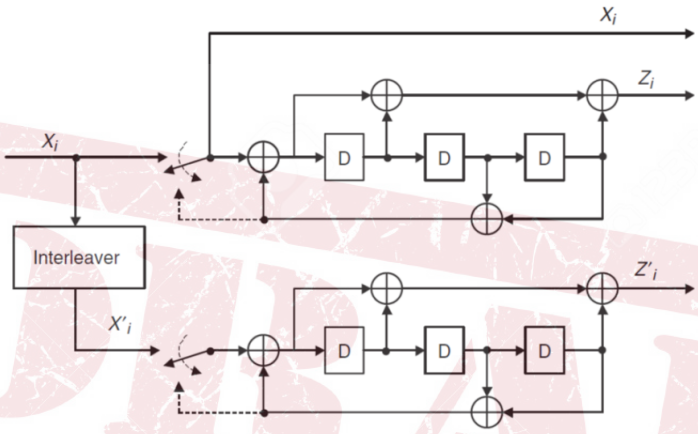


Figure 5.16 The rate-1/3 3gpp Turbo Encoder

When the data block is to be sent out on the channel, we assume as usual that the logic binary symbols $a[n]$ with alphabet (0,1) are re-mapped onto antipodal symbols $a_\pm[n]$ (-1,+1), respectively. The coded symbols are corrupted by AWGN, so that, with the usual notation, the received signal samples block is $\mathbf{r} = \mathbf{a}_\pm + \mathbf{w}$.

5.3.2 Iterative Decoding of the Turbo Code

The main components of the turbo decoder in Fig. 5.17 are the two Soft-Input, Soft-Output BCJR decoders for the two constituent RSC's. In particular, the received block $\mathbf{r}$ is demultiplexed into the three components $\mathbf{r_b}$, $\mathbf{r_p}$, $\mathbf{r}_\mathbf{p}^\pi$ that represent corrupted versions of $\mathbf{b}$, $\mathbf{p}$, and $\mathbf{p}_\pi$, respectively (after re-mapping). As is seen, $\mathbf{r_b}$ and $\mathbf{r_p}$ are used by the first SISO decoder DEC1 to compute LAPPRs values $L^{(1)}(b[k])$ according to the BCJR algorithm, and also to produce an a-posteriori value $L^{(1)}_{out}(b[k])$ that is routed towards the second decoder DEC2, together with permuted $\mathbf{r_b}$ (that we call $\mathbf{r}_\mathbf{b}^\pi$) and the samples relevant to the respective parity bits $\mathbf{r}_\mathbf{p}^\pi$. The key issue is that DEC2 receives quantity $L^{(1)}_{out}(b[k])$ that was computed by DEC1 using (noisy) parity bits $\mathbf{r_p}$ that are *extrinsic* (i.e., external) to the process of coding of RSC1: this is just the extrinsic information (what we called a "second opinion") we sought for. DEC2 uses this information as an *input* extrinsic information $L^{(2)}_{in}(b[k])$, together with the other inputs, to derive its own version of the extrinsic information $L^{(2)}_{out}(b[k])$ that can be

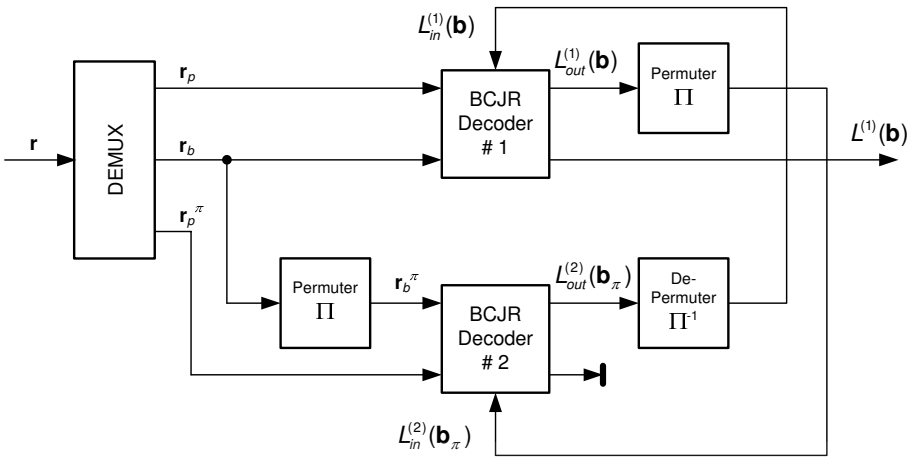


Figure 5.17 PCCC Turbo decoder (b)

routed back towards DEC1 since it is *extrinsic* to it (it is computed using $\mathbf{r}_{\mathbf{p}}^{\pi}$, which DEC1 does not have access to). It makes sense then trying to further refine $L^{(1)}\,(b[k])$ using this new piece of extrinsic information (that we interpret as input information $L_{in}^{(1)}(b[k])$), and also to provide a new, further updated and improved extrinsic information to be handed over again to DEC2...

We have just described the two-stage iterative algorithm with feedback information propagation that we call *turbo decoding*. This spinning of extrinsic information between the two decoders recalls the fast spinning of the turbine in a turbo-compressed engine, that attains the very same boost on the engine performance as we attain here in channel decoding. After a (small) number of iterations, we see that the values of the LAPPR (and the extrinsic information values) do not appreciably change any longer, and iterations are stopped. The final (hard) decision on $b[k]$ is taken indifferently on any one of the two LAPPRs from DEC1 or DEC2 (in our case, L_1). Appendix A specifies in detail the difference between the whole LAPPR L that is used for final bit estimation, and its extrinsic part L_e that is propagated between the two decoders.

Admittedly, turbo decoding is not optimal for the PCCC encoder in Fig. 5.15. We may wonder then how to implement an optimum decoder, and how far its performance would be. Concerning the first question, we recognize that, after all, the turbo encoder is nothing but a peculiar finite-state machine, whose evolution can be represented by a trellis, and whose input string can be decoded from a noise-corrupted output by a suited Viterbi Detector. How many states does the machine have, and so how many nodes does the trellis require? We see that the *memory* of the encoder is substantially concentrated in the permuter. We can regard the permuter as a large random-access memory that is written in a natural order and is read according to a different pseudo-random sequence. This is the just way it is implemented in hardware. So the memory size is N, and the number of states is $N_s = 2^N$. We already know that N can be as large as 10^4, so that N_s becomes a meaningless number, and the optimum decoder is a chimera. In addition to this, we'll see in a while that the room for improvement of this hypothetical decoder is not so large, since the performance of the (suboptimal!) turbo decoder is anyway remarkably close to the Shannon limit.

We may also wonder how to design turbo encoders with rates different from the "native" 1/3 of in Fig. 5.15 (a). What if we intend to *increase* the rate to, say, 1/2? The most simple

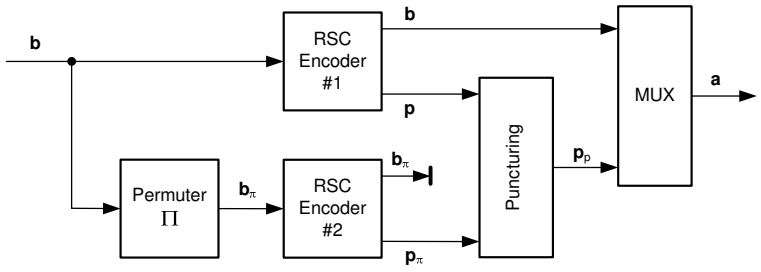


Figure 5.18 PCCC rate-1/2 encoder with puncturing

$p[0]$ • $p[2]$ • $p[4]$ • $p[6]$ • ...
• $p_\pi[1]$ • $p_\pi[3]$ • $p_\pi[5]$ • $p_\pi[7]$ •

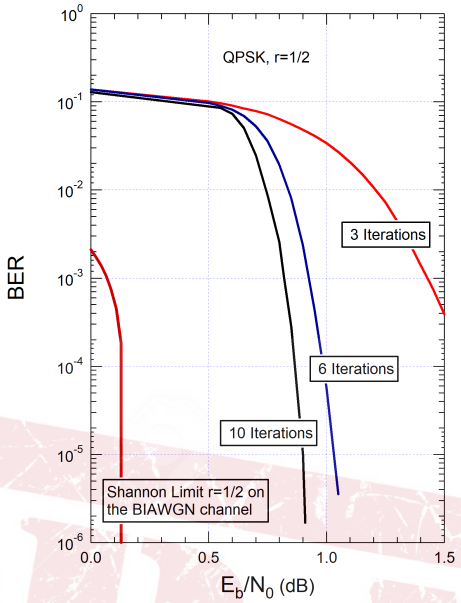
Figure 5.19 Puncturing pattern of the PCCC rate-1/2 encoder

and most practiced solution is shown in Fig. 5.18. The only difference with the original scheme is the insertion of a *puncturer* that modifies $\mathbf{p}$ and $\mathbf{p}_\pi$. The output parity block $\mathbf{p}_p$ (where of course the subscript stands for *punctured*) is obtained as depicted in Fig. 5.19, i.e., by interleaving the even-numbered components of $\mathbf{p}$ with the odd-numbered elements of $\mathbf{p}_\pi$ and puncturing (i.e., eliminating) the others. The resulting $\mathbf{p}_p$ has size K as well, so that the overall rate is 1/2. In the decoder, the missing (punctured) elements of $\mathbf{p}$ and $\mathbf{p}_\pi$ are restored as 0s, so that two size-K parity blocks are anyway available for decoding. In spite of the puncturing, that causes suppression of some redundant bits, the decoder still has remarkably good performance, of course no better than the original code because of the increase of the rate.

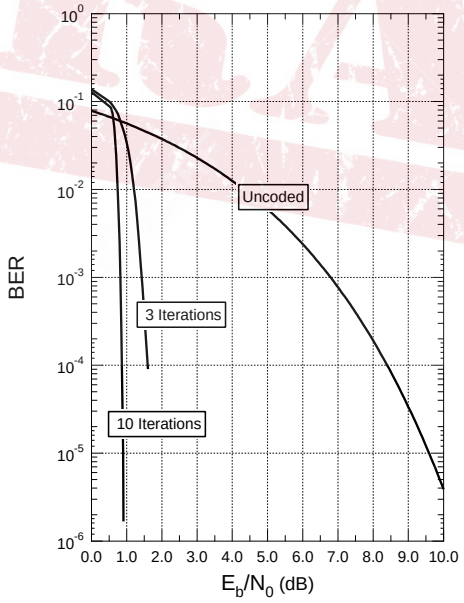
5.3.3 BER performance

The boost to decoding performance achieved with turbo decoding is impressive. Fig. 5.20 (a) shows the BER curves obtained by simulation of a rate-1/3 turbo code with a pseudo-random permuter/interleaver with size $K = 1500$ bits and no optimization whatsoever. Major performance improvements are attained with the first few iterations, then the performance increase gets minor. Also impressive is the comparison with uncoded trasmission (Fig. 5.20 (b)) that reveals a coding gain @ 10^{-5} of almost 9 dB. For reference, we have also reported the limit BER curve deriving from the so-called *converse* Shannon capacity theorem. We see that this "off the shelf" code with a relatively small interleaver lies at less than 1 dB from the Shannon limit.

When we speak of decoding iterations, it should be clear from our description of the turbo algorithm above that all iterations are carried out in virtual time by "replaying" the same data $\mathbf{r_b}$, $\mathbf{r_p}$, $\mathbf{r_p^\pi}$ for the same N-bit code block into the decoder, until convergence is attained. If we take as a typical number of iterations the figure $N_{it} = 10$, we can evaluate the complexity of turbo decoding with respect to conventional Viterbi decoding of the constituent RSC code by considering that: i) the complexity of each BCJR SISO detector is roughly double that of a VD; ii) each decoding iteration calls for the execution of two BCJRs ; iii) we do 10 iterations. Therefore, the complexity of a turbo decoder is roughly $2 \cdot 2 \cdot 10 = 40$ times the VD's.



(a)



(b)

Figure 5.20 BER curves of a turbo decoder with different numbers of decoding iterations (a) and compared to uncoded transmission (b)

5.4 Low-Density Parity-Check Codes and the Iterative Message-Passing Decoder

5.4.1 What is an LDPC code?

The new notion of iterative detection of channel codes based on probabilistic SISO decoding spurred a fresh reconsideration of old codes and decoding problems left unsolved. In the 60s, Gallager invented a class of parity-check codes (much like the Hamming code of Section 5.1.1) that were very peculiar for the time, since they were not designed following the paradigm of algebraic codes. Nonetheless, they were shown to possess very good Hamming distance properties, especially for large block size. Gallager, following what shown by Shannon in his celebrated proof of the capacity theorem, focused on certain kinds of randomly generated parity-check matrices, and computed the minimum Hamming distance of the respective codes for increasing matrix size, showing that d_{min} grows with N with unit probability, so that any "large" code is very good. He also outlined an algorithm to perform what he called "probabilistic decoding" that went basically unnoticed because impractical to implement. In the 90s, MacKay re-discovered such codes, extended the construction of efficient codes with other criteria than Gallager's, re-formulated the iterative decoding algorithm, and showed that such codes came really close to Shannon capacity.

An LDPC code is simply a parity-check code whose binary parity check matrix $\mathbf{H}$ is i) large, ii) *sparse*, iii) randomly generated. "Sparse" means that it is mainly made of 0 entries, so that the entries 1 are very rare - low density! $\mathbf{H}$ looks like the sample that follows for a

(50,25), $r = 1/2$ code:

$$\mathbf{H} = \begin{bmatrix}
0000000010000000000000000000000100010001100001000 \\
0000000000000000000000000001000001000000100010011 \\
0010000100000000001000000001100100000000000000000 \\
0000010000000000010001000000000000001100000000010 \\
0100000100000000000001000000100000010000100000000 \\
0000001000001000000000100100000100100000000000000 \\
0000000001010010100000000000000000000100100000000 \\
1000001000000100000100000000000000000000000100100 \\
0000000000000000100001000000000000010000010100100 \\
1000000000010000001010000000000100000100000000000 \\
0000000010000100100010000001001000000000000000000 \\
0100000000100010001000000000001000000000010000000 \\
0000010000000001000000000010100000000000010010000 \\
0001000000000000000000100000000010010000000011000 \\
0010000000001010100000000001000100000000000000000 \\
0000000000000000000001010001000000000000000000001101 \\
0000000100000000000000001100000010000100001000000 \\
0000100010011000000101000000000000000000000000000 \\
0000100000100000000000100000100101000000000000000 \\
0100101001000000000000000000000000000010000000010 \\
0000000000000100100000010001000000000000001000100 \\
0000000000001001000000100000000000000010000110000 \\
1001010000000000000000000000001000110000000000000 \\
0000000001100000000100001000000000000000100000001 \\
0011000000000000000000001100000000000110000000000
\end{bmatrix}$$

You can see why an LDPC matrix gives rise to a " good " code: if the rows (columns) of the matrix consist of almost all 0s, the number of rows (columns) you need to add to give a vector $\mathbf{0}$ (and this number is just the d_{min} of the code) is high because the 1s are sporadic, and so to "delete " them from the sum vector, you have to find another row (column) with a 1 in the same position - an event with a very low probability that requires consideration of *many* rows (columns) - hence a large minimum distance. We will see that good LDPC codes are based on large parity check matrices (up to a size of 10^5), so that, in addition to the (usual) issue of of finding efficient decoding algorithms, such codes also need efficient *encoding* procedures. In general, the encoding operation **a=bG** of a block code has quadratic complexity with K, which for large size becomes impractical. In addition, the code has always to be for practical reasons *systematic*. The fact that $\mathbf{H}$ is low-density helps very much in the decoding procedure, as we will see later. For encoding, observe that $\mathbf{H}$ is totally unstructured because it is randomly generated, and so it is easy to see that the generator matrix $\mathbf{G}$ in systematic form (after Gaussian elimination) is *not* low-density, making the complexity of encoding high again.

In our random construction of $\mathbf{H}$, the only constraint to be satisfied is 'e just the density (average or exact) of the elements equal to 1. An LDPC code is *regular* if the number of 1s in any row of $\mathbf{H}$ w_r is always the same (Gallager's codes), as is the number of 1s on

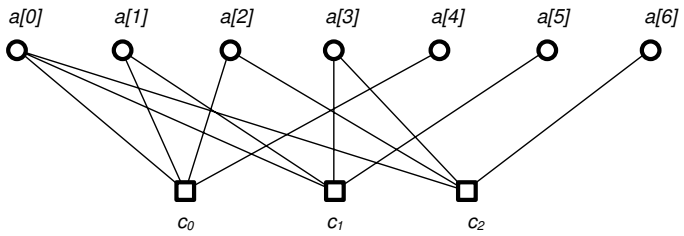


Figure 5.21 Tanner graph of the parity-check matrix (5.50)

each column w_c. If it's not regular, the code is said to be *irregular*. Our matrix (5.50) is representative of an irregular code. We have good design rules for both regular and irregular codes. In general, regular codes (like Gallager's) give rise to efficient encoding rules that, based on the regularity of the sparse matrix, allow real-time encoding.

In the 80s, still to solve problems related to efficient code design, Tanner introduced a different representation of a parity-check code that reveals particularly useful in the discovery of good LDPC codes, and in the design of efficient decoders. We know that each row of the parity check matrix represents a check equation that is satisfied by the symbols of a generic codeword $\mathbf{a}$. We represent this by the so-called *Tanner graph* associated to $\mathbf{H}$, whose construction we show with a more manageable example than the matrix above. Assume we have the usual $(N = 7, K = 4)$ Hamming code with parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{5.50}$$

We build a *bipartite* graph, i.e, a graph with two different kind of nodes that are connected by edges (lines). One set of N nodes (the so-called *bit nodes* or *variable nodes* $a[n], n = 1, \ldots, N$) that we represent as circles in a row, stand for the values of the codeword symbols $a[n]$ (as the notation directly suggests). The other set of $N - K$ nodes (the *check nodes* $c_i, i = 1, \ldots, N - K$)) represent the parity-check equations, and we draw them as another row of squares below the bit nodes row. Then, to represent the particular structure of $\mathbf{H}$, and also of the parity-check equations, we connect to each check node the relevant bit nodes that take part in the corresponding parity-check equation. In other words, and more formally, a check node c_i is connected to a bit node a_n iff the entry $h_{i,n}$ of $\mathbf{H}$ is equal to 1. The resulting Tanner graph is shown in Fig. 5.21. How this can be useful to code design and decoding, we'll see in the next subsections. We only mention here that the Tanner graph (whose edges are low density...) of a good LDPC code does not have *short cycles*. A cycle in the graph is a path that starts on one node and comes back to the same node after a number of passages through (distinct) edges and nodes. The minimum cycle length that is found on a certain graph (i.e., the minimum number of edges on the path) is called its *girth* and has to be as large as possible, for sure larger than the bare minimum that is (as is apparent) 4. We will not insist on code design which, as with turbo codes, is very complicated, and we will move on to the explanation of iterative decoding. We just mention here that both turbo and LDPC codes meet the notion of *random code with large blocklength* introduced by Shannon in his proof of the capacity theorem.

Of course, decoding a truly random code with large blocklength is virtually impossible even with the best of today's technology: the only way is exhaustive distance evaluation to

implement a minimum-distance decoder. In the past, the design of a good code relied on the discovery of suited codebooks with nice algebraic properties (think of our Hamming code) that ensured good distance properties on one side, and easy decodability on the other. Many design rules were discovered that led to different families of block codes (Fire, Bose-Chauduri-Hocquengheim or BCH, Reed-Solomon, cyclic codes and so on). They all were based on strongly structured codebooks or design rules. On the contrary, turbo codes and LDPC codes resemble Shannon's random codes in that both exhibit an inherently random feature: the permuter and the matrix $\mathbf{H}$, respectively. Therefore, their decoding algorithms are not straightforward and have to be invented and tailored (also) through empirical criteria.

5.4.2 Iterative decoding of LDPC codes - I: Hard-Input-Decoding

The Tanner graph is the basis to explain how an iterative algorithm to decode an LDPC code can be implemented. Assume that our input is the binary vector $\mathbf{d} = [d[0], d[1], ..., d[N-1]]$ at the output of a BSC after transmission of a codeword $\mathbf{a}$. This vector represents our first, initial guess of the decoded codeword, that we will *refine* with an iterative procedure. To mean this, we let $\mathbf{a}^{(0)} \stackrel{\triangle}{=} \mathbf{d}$.

In the Tanner graph, the nodes have to be regarded as local processors that receive inputs along the edges they're connected to, make computations, and outputs the results of such computations onto the same edges. The initial binary values above represent the first set of "messages" sent by the bit nodes down to the check nodes along the edges of the graph (first half-iteration). The check nodes receive such messages and perform one action: they all verify their own parity equation, by just adding all the messages they receive, getting 0 or 1 each. If all results by chceknodes are 0, then all parity check equations are satisfied, and decoding stops with the current values of the bit nodes, that represent the decoded codeword $\hat{\mathbf{a}}$. If on the contrary (as in Fig. 5.22a) some result is 1, this is an indication that some bit node is wrong and has to be modified, so that another half-iteration is started by sending messages up to the bitnodes. Figure 5.22 helps visualizing the two half-iterations we have just described. Referring to the usual sample code with parity check matrix (5.50), assume that the transmitted codeword is $\mathbf{a} = [1011001]$, and that the output of the channel gives $\mathbf{d} = [1\mathbf{1}111001]$. The situation after the first two half-iterations is depicted in Fig. 5.22a, where we indicate the values computed by the check nodes.

After the check node messages are received, the bit nodes have to perform some action in turn. Each bit node has to apply some criterion to possibly change (flip) its own value so as to "improve" decoding. A good rule for flipping is: "invert the current value of the bit if the number of received messages equal to 1 (number of connected failed check nodes) is larger than t", with the rationale that in this case too many checks fail to assume that the current value is correct, so that it has to be flipped. Form this standpoint, the message received by a bit node is interpreted like a "wrong bit" flag provided by the checknode. The value of t is a design parameter that affects the speed of convergence of the iterative algorithms. Assuming $t = 1$, what we get after flipping is the situation in Fig. 5.22b, wherein we have the new current estimate of the codeword $\mathbf{a}^{(1)}$ written into the bit nodes. We need another semi-iteration towards the checknodes to verify if this is the final codeword, but we discover (not shown in the figure) that still some check node fails, and so on.

Decoding is stopped either when all checks are satisfied, as stated above, or when a pre-set maximum number of iterations N_{it} is reached. Retrieving the source (information) bits $\hat{b}[k]$ from the decoded bits $\hat{a}[n]$ after N_{it} iterations is trivial since we are using systematic

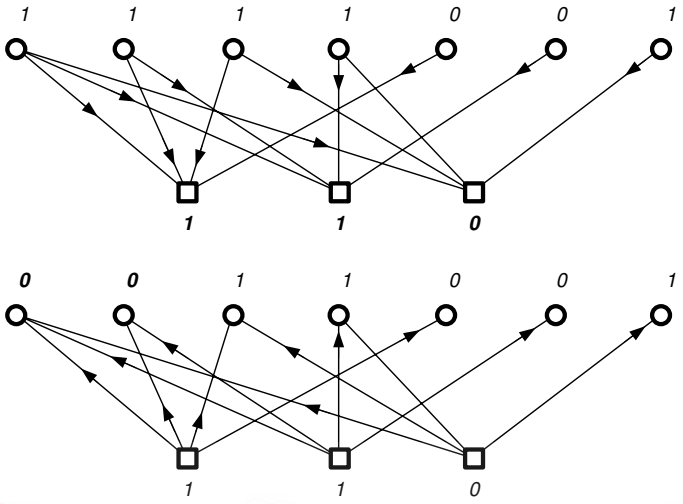


Figure 5.22 Example of iterative decoding with message passing. After checknode (a) and after bitnodes(b) computations

codes: $\hat{b}[k] = \hat{a}[k], k = 0, \ldots, K-1$. There is no guarantee that this iterative procedure, called *bit-flipping* algorithm, is optimum, neither that it leads to a "good" codeword estimate. Nonetheless, extensive experimental results suggest that the procedure yields low BER in the vast majority of practical cases.

5.4.3 Iterative decoding of LDPC codes - II: Soft-Input-Decoding

What if we have available the soft output of the channel $\mathbf{r}$ instead of the hard-detected codeword symbols $\mathbf{d}$ as in the discussion above ? We have to develop a new iterative message-passing algorithm wherein all messages that travel on the Tanner graph edges are now *soft*.

The aim of the SISO decoder is not only finding (final) estimated binary values $\hat{a}[n]$for each $a[n]$, but also providing the probabilities

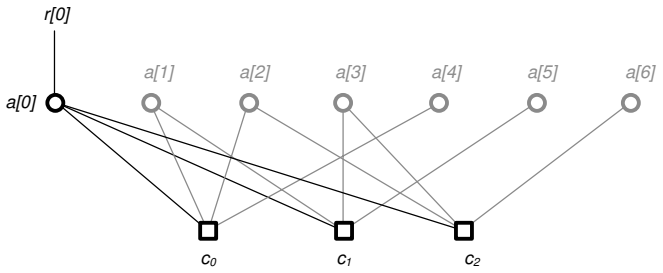
$$\Pr\left\{a[n] = x|\mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\} \quad x = 0, 1 \tag{5.51}$$

or equivalently the LAPPRs

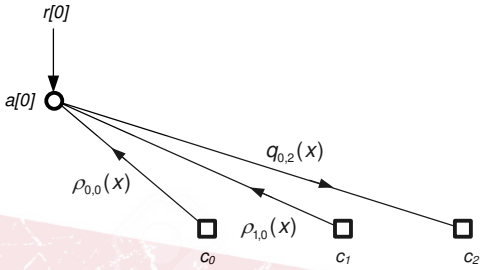
$$L(a[n]) \triangleq \ln\left(\frac{\Pr\left\{a[n] = 1|\mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}}{\Pr\left\{a[n] = 0|\mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}}\right) \tag{5.52}$$

where $C_{i,n}$ is the set of indices i of the parity checks that involve bit $a[n]$ (this means that *all* parity checks containing $a[n]$ have to be satisfied).

The question is, how can we derive such quantities based on the iterative exchange of messages on the Tanner graph? As with turbo codes, we will leave the detailed derivation and description of the resulting message-passing algorithm to Appendix B, giving here the flavor and the general outline of the decoding process. We start by remarking that the soft-input algorithm follows very much the procedure we have described in the previous



(a)



(b)

Figure 5.23 Subgraph for $a[0]$ of Fig. 5.21 (a) with downward exchanged messages (b)

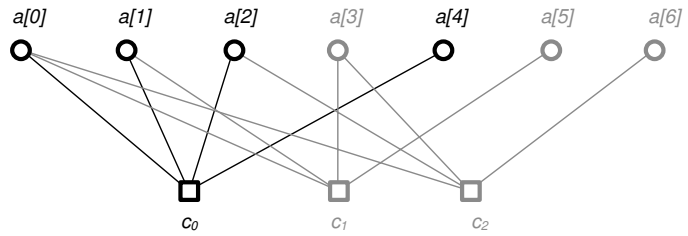
subsection for hard decoding. Of course, the messages that are interchanged in the two half-iterations are different. They are soft messages that give to each node (check or variable) some piece of *extrinsic information* the node did not have before.

Consider again our familiar (7,4) Hamming code, and let us isolate the *subgraph* relative to the first bitnode $a[0]$ (the first column), as in Fig. 5.23 (a). We have added a further edge to mean that the soft channel output $r[0]$ also sends information to the bitnode $a[0]$.

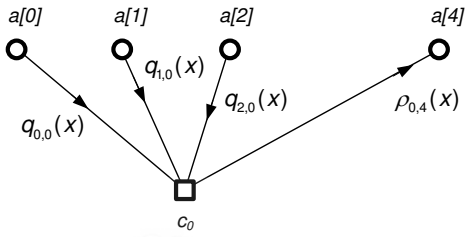
Let us now consider the soft messages that flow on the edges of the graph, and let us denote by $q_{n,i}$ the message that is sent (downwards) from the bit node n to the control node i, and with $\rho_{i,n}$ the message (upwards) sent by the check node i to the bit node n, $n = 0,\ dots, N-1, i = 0,\ dots, N-K-1$. We take as a particular example the message from the bit node 0 to check node 2, that is, $q_{0,2}$ as illustrated in Fig. 5.23 (b). In the figure, the arrows clearly indicate the flow of messages on the edges, suggest the type of calculation that the node performs, and also address the extrinsicity issue: the node c_2 receives a message from $a[0]$ calculated from the relevant inputs, that is, $r[0]$ and the check node messages received upwards from $a[0]$ itself in the previous iteration, i.e., $\rho_{i,0}$. Amidst the latter, $rho_{2,0}$ does *not* appear, because the information it would bring is not extrinsic to $a[0]$, as it was calculated with a value of $a[0]$ from the previous iteration which node $a[0]$ already knows!

A similar situation occurs considering the messages sent by the bit nodes to the check nodes as in Fig. 5.24 for $rho_{0,4}$, where the check node does *not* receive $q_{4,0}$ because it is not extrinsic. In the figures, the messages appear to depend on a variable x: theoretically, each edge does not carry *just one* value, rather it conveys *both probabilities* of certain events, depending on the assumed value of the bit $a[n] = x$, $x = 0, 1$. Specifically,

$$q_{n,i}(x) \stackrel{\triangle}{=} \Pr\left\{a[n] = x \,\middle|\, r[n], \{\rho_{i',n}\}_{i' \neq i}\right\} \quad x = 0, 1 \tag{5.53}$$



(a)



(b)

Figure 5.24 Subgraph for c_0 of Fig. 5.21 (a) with upward exchanged messages (b)

and

$$\rho_{i,n}(x) \triangleq \Pr\left\{c_i = 0 \,\middle|\, a[n] = x, \{q_{n,i'}\}_{i' \neq i}\right\} \quad x = 0, 1 \tag{5.54}$$

We said above "theoretically" concerning the probabilistic messages, because what actually the algorithm computes and exchanges are the LAPPRs of the messages above:

$$L(q_{n,i}) \triangleq \ln \frac{q_{n,i}(1)}{q_{n,i}(0)} \quad , \quad L(\rho_{i,n}) \triangleq \ln \frac{\rho_{i,n}(1)}{\rho_{i,n}(0)} \tag{5.55}$$

We are already familiar with the notion of LAPPR $L(b[k])$ related to the value of an information bit. In the message-passing algorithm the message $L(q_{n,i})$ is something similar: it is the "amount of confidence" or *belief* that each bit node can give to the check node about the value of that bit - not only its binary value - to enable further computation of the check node itself (in the bit-flipping version, to compute the parity-check). Also, the message $L(\rho_{i,n})$ is the "amount of confidence" or *belief* that the check node can give to the bit node about the value of the bit, so as to make its own parity equation satisfied, and possibly induce a change in the bit estimate (in the bit-flipping version, possibly flip a bit). This is why the soft-message passing algorithm is also called *belief propagation*: the messages that are interchanged are not hard, binary decisions on bit values. Rather, they are soft-values *beliefs* about the same quantities.

A nice feature of the message-passing LDPC decoder is the amenability to an easy VLSI implementation, following the idea of the Tanner graph, which suggest a parallel computing stucture. Each node becomes an elementary processor bearing the feature of local computation that is implicit in parallel architectures. Processors of a certain class (bit, check) receive at each half-iteration a set of soft inputs, and compute *in parallel* the respective soft output values to be sent to the other class of processors, and so on. The resulting hardware structure is regular, efficient, and simple to design, as opposed to the BCJR decoder for turbo codes which turns out to be more complicated from the standpoint of hardware implementation.

The specific computation for each node class is derived in Appendix B, but turns out to be real elementary. For the bitnode-to-checknode message at iteration $m+1$ we have:

$$L^{(m+1)}(q_{n,i}) = \frac{4E_s}{N_0} r[n] + \sum_{i \in C_{i,n}, i \neq n} L^{(m)}(\rho_{i,n}) \tag{5.56}$$

We may call this equation a sort of ''soft majority decision" on bit $a[n]$: the first term is the soft output of the channel that we may use to decode $a[n]$ without actually using any channel code information; the second term is the sum of additional information terms (beliefs) on the bit provided by the checknodes, that may change the sign of the overall decision, thus correcting possible errors. The more beliefs are in agreement (i.e., of the same sign...), the more likely is that the sign overall belief on the decoded bit is correct.

For the checknode-to-bitnode message, the expression is formally more complicated, but equally simple to understand and interpret, at least in the so-called *min-sum* simplified implementation of the decoder:

$$L^{(m+1)}(\rho_{i,n}) = - \left\{ \prod_{n' \in R_{i \setminus n}} \operatorname{sgn}[-L^{(m)}(q_{n',i})] \right\} \cdot \min_{n' \in R_{i \setminus n}} |L^{(m)}(q_{n',i})| \tag{5.57}$$

The rationale of this computation is twofold: the the sign of (5.57) is such that the corresponding parity-check equation is satisfied, or in other words the event $c_i = 0$ in (5.54) is verified - to understand this, observe that in our mapping of logica values 0 and 1 to -1 and +1 on the channel, the operation of modulo-2 sum $a \oplus b$ is translated into $-(-a_{\pm} \cdot -b_{\pm})$. For the modulus, the *reliability* of the computed check is considered to be the weakest of the different reliabilities of the messages forming the check (kind of the weakest ring of the chain).

At each decoding iteration, the LAPPR on $a[n]$ is computed and hard-detected to derive a tentative decoded codeword binary value $\mathbf{a}^{(m)}$. If the latter satisfies all parity checks, decoding is stopped, and $\hat{\mathbf{a}}^{(m)} = \mathbf{a}^{(m)}$. If it does not, more iterations follow until a maximum number N_{it} is reached, so that a final decision is anyway performed. We still have to say how the LAPPR $L(a[n])$ as in (5.52) is derived. Appendix B shows that

$$L^{(m)}(a[n]) = \frac{4E_s}{N_0} r[n] + \sum_{i \in C_{i,n}} L^{(m)}(\rho_{n,i}) \tag{5.58}$$

We can see that the LAPPR we sought for is composed of a channel information term, independent of the particular code (see also Example 31), plus the result of the execution of iterative (soft) message-passing. As with turbo codes, we see that at each decoding iteration the LAPPRs of the codeword symbols change their values and at times change sign, so that the corresponding hard detections are possibly changed. This makes the BER improve, until decoding stops.

The complexity of the decoding algorithm depends on the number of messages which are exchanged (and therefore must be calculated) on the graph. If matrix $\mathbf{H}$ were random with equiprobable binary elements, each bit node has an average number of edges equal to $(N-K)/2$, and the complexity of the decoder would be $N(N-K)$, i.e., *quadratic* with the code order. If, on the other hand, the matrix is low-density, and the code is for simplicity regular, w_c edges are connected to each bit node, with $w_c \ll n$ (otherwise the code is *not* low density) and the complexity of decoding is now Nw_c, i.e., *linear* with the order of the code!

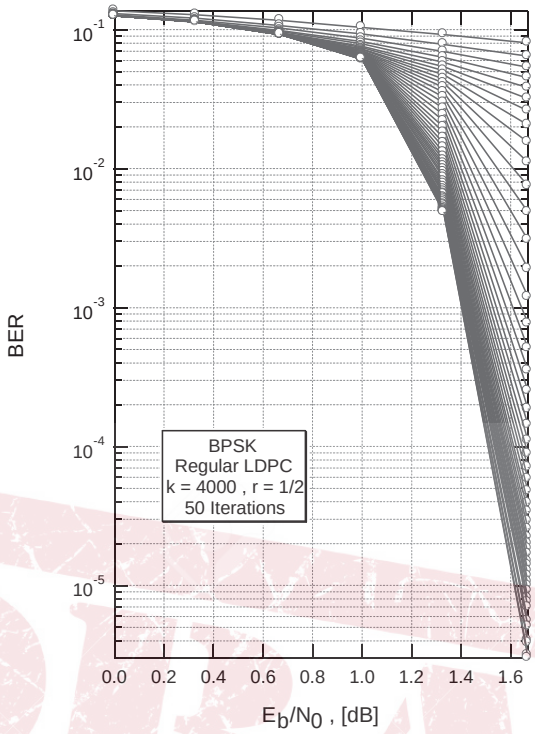


Figure 5.25 BER of an LDPC code with different number of decoding iterations

As a didactic example, we see in Fig. 5.25 the error rate on the detected bits $b[k] = \hat{a}^{N_{it}}[k]$ of a (8000,4000) systematic regular LDPC on the AWGN channel with binary antipodal signaling. The improvement in the BER for increasing number of decoding iterations is apparent. What is not seen in the picture is an annoying phenomenon of message-passing decoders: when the BER is smaller (smaller thatn is seen in Fig. reffigldpc1), the slope of the BER curve for increasing E_s/N_0 changes, as it tends to roll-off more "horizontally" then before, almost reaching a BER "floor". In applications requiring very low bit rates, like high-quality video that calls for 10^{-10} at least, this may represent a real disadvantage of the otherwise well-performing code. The countermeasure to this is something already well known in video transmission: equipping the (*inner*) LDPC with an *outer* code that corrects the residual errors produced by the message-passing decoder and compensates for the floor. This is exactly what is done in the channel code for DVB-S2 video transmission, where an LDPC with long blocks ($N = 64800$) is concatenated with an outer very-high-rate ($r \simeq 0.994$) (32400,32208) BCH algebraic code. The impressive performance of the resulting concatenated LDPC/BCH code is presented in Fig. 5.26 that shows the BER of the different constellations of the DVB-S2 satellite television standard (QPSK,8PSK,16APSK,32APSK) with $N = 64800$ and with different overall rates $r = r_i r_o$.

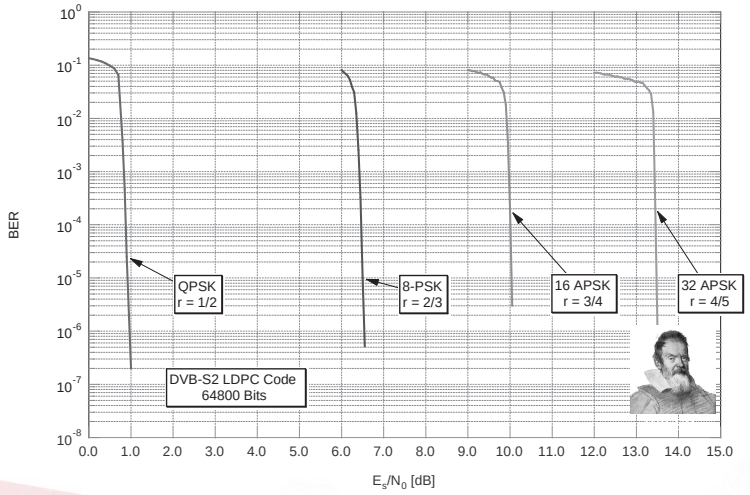


Figure 5.26 BER of the standard LDPC code for DVB-S2. Courtesy of WISER srl

5.5 Polar Codes with Successive-Cancelation Decoding

5.5.1 Do we really need Yet-Another-Coding-Scheme?

The spectacular success of Turbo- and LDPC-Coding with iterative decoding almost set an end to research in new coding schemes. After all, the best known LDPC codes, very effective and simple to implement like those used in DVB-S2, were less than 0.8 dB away from Shannon capacity. But, this spectacular performance comes only for a sufficiently high block-length - for DVB-S2 $N = 64800$. Large blocks turn out to be prohibitive in real time or anyway latency-constrained communications (two-way voice, two-way video, machine-to-machine communications etc.). Optimized short-block communications was still an issue in the late 2000's after the advent of iterative decoding. Take for instance the 3gpp PCCC turbo code depicted in Fig. 5.16, whose (simulated) BER performance is shown in Fig. 5.27 for variable blocklengths: It is seen that the coding gain is *strongly* dependent on the block size. On the other hand, this is something to be expected: the Shannon capacity curve that we used in Fig. 5.38 represents the *unconstrained* capacity, i.e., the performance limit for infinite-length block codes. When the blocklength is actually finite, channel capacity is reduced, and a certain penalty is to be expected from *any* code. Figure 5.28 shows a lower (red) and an upper (dash-blue) bound of the penalty in terms of SNR E_b/N_0 with respect of the unconstrained Shannon capacity for a finite blocklength K. For blocks as short as some hundred bits, a degradation of a couple of dB's is to be expected. And, channel codes as efficient as (long-block) LDPC were not available.

The need was therefore: finding close-to-capacity codes for short blocks - the subject of the next section.

5.5.2 Polarization Theorem and Polar Codes

Polar codes were invented by Arikan in 2009. They are based on the idea of *polarization* of a set of parallel communication channels. Why parallel? If the N consecutive uses of a

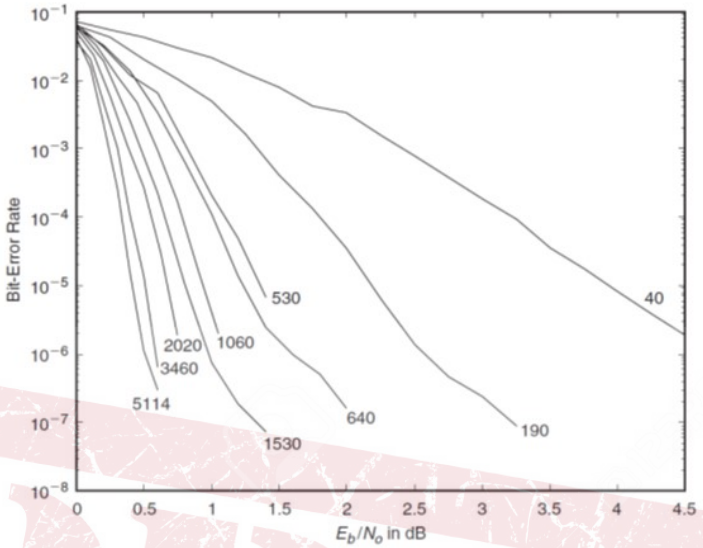


Figure 5.27 BER of the standard PCCC Turbo Code of 3gpp

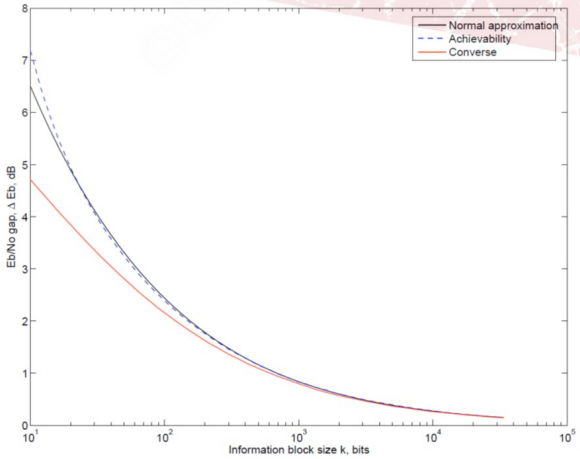


Figure 5.28 Penalty of finite-length block codes wrt unconstrained Shannon capacity

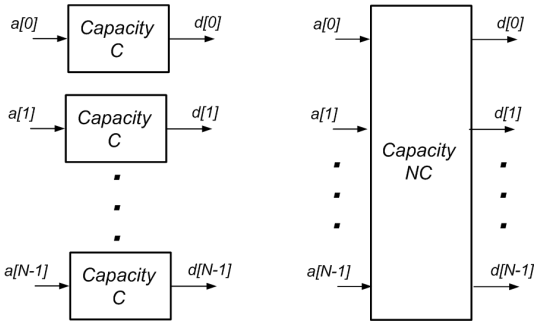


Figure 5.29 Vector Channel as an aggregation of Scalar Channels

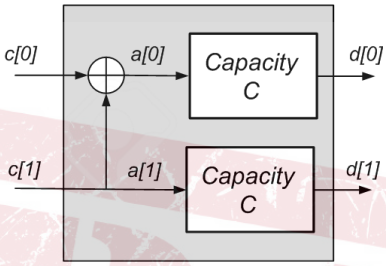


Figure 5.30 2×2 Polar Precoding

noisy channel to send out the N binary symbol of a code block $\mathbf{a}$ one after the other are actually independent, we can regard transmission of the whole block as taking place on a *parallel* channel of size N whose output is the channel block $\mathbf{d}$. The total Shannon capacity of the channel will be $\mathcal{C}_{vec} = N\mathcal{C}$ where $\mathcal{C}$ is the capacity of the original "scalar" channel in bits/channel use. To simplify, we can regard the scalar channels as simple, symmetric BSC or BEC channels, both with the property that capacity equals the input-output mutual information in the case of symmetric input probability distribution. This vector channel is actually an artifact since the parallel scalar channels are independent of each other, and their aggregation to form a vector channel does not actually add anything to the problem.

The fundamental idea of polar coding is now to "redistribute" capacity by creating a *different* set of information-transmitting channels by appropriate (pre-)coding of the information bits. The total capacity of such new channels will be unchanged, i.e., still equal to $\mathcal{C}_{vec}$ because pre-coding will be a deterministic, invertible transformation, but the individual capacities of the new bitwise channels will be (much) different from each other, with many "good" channels with capacity close to 1, and a few "bad" channels with capacity close to 0 - this is called *polarization* of the transmission channels.

Let us explain this general principle with a simple 2×2 example. Figure 5.30 shows a simple way of precoding two inputs bits $c[0]$ and $c[1]$ to obtain two coded bits $a[0]$ and $a[1]$ to be sent into two parallel and equivalent channels whose output is $d[0]$ and $d[1]$, respectively (they may be two independent BSC's). The two new information transmitting channels are *not* to be intended single-input single-output (i.e., $c[0]$ to $d[0]$ and separately $c[1]$ to $d[1]$) as before: channel #0 is between $c[0]$ and *the whole received vector* $\mathbf{d} = [d[0], d[1]]$, while channel #1 is between $c[1]$ and $(\mathbf{d}, c[0])$ where concatenation of $\mathbf{d}$ with $c[0]$ means that decoding of $c[1]$ also exploits the previous detection of $c[0]$. The capacity of the new vector

channel will still be $\mathcal{C}_0 + \mathcal{C}_1 = 2\mathcal{C}$, but now it is clear that $\mathcal{C}_1 \geq \mathcal{C}$ because now channel 1 incorporates the observation of additional outputs ($d[0]$ and $c[0]$) with respect to the case of previous $c[1]$-to-$d[1]$ scalar channel, and so the mutual information is in general no smaller than before. As a consequence, $\mathcal{C}_0 \leq \mathcal{C}$ or in other words $\mathcal{C}_0 \leq \mathcal{C} \leq \mathcal{C}_1$: we have obtained two different channels, one better than the other, i.e., *polarized*. As a special case, if the original scalar channels are two independent and equal BEC(p_{CE}) channels whose capacity is $\mathcal{C} = 1 - p_{CE}$, it is found that

$$\mathcal{C}_0 = (1 - p_{CE})^2 < 1 - p_{CE} < 1 - p_{CE}^2 = \mathcal{C}_1 \tag{5.59}$$

The simple 2×2 case study before already suggests a naive rate-1/2 encoding scheme: the idea is *not* to use the "bad" new channel #0, only using the "good" new one #1. This can be easily done by *freezing* $[0]$ to a known, fixed value, for instance 0, and using only the other input bit $c[1]$ as the actual information bit to be sent out on a better communication channel. The decoder knows that $c[0]$ is 0 and uses this piece of information to perform optimum decoding of $c[1]$ observing $\mathbf{d}$.

No wonder that actual polar codes are more complicated than this. First, the construction above has to be generalized in order to stretch polarization to its extreme: out of a length-N input block, create K bit channels whose capacity will be very close to 1, with reliable communication, paired with $N - K$ additional channels whose capacity will be close to 0, implicitly used to "convey" frozen bits. The rate of the code will be in this way $r = K/N$ as ever, and the fundamental *polarization theorem* will apply:

When N is sufficiently large, for any (arbitrarily small) ε the number of polarized channels, out of the N original ones, whose capacity is larger than 1-ε is equal to $N\mathcal{C}$, so that the number of channels whose capacity is smaller than ε is $N - N\mathcal{C}$

This guarantees that the rate of the (large block) code $r = K/N = (N\mathcal{C})/N$ is asymptotically equal to the capacity $\mathcal{C}$ of the original scalar channel. The polar codes are the only practical codes whose asymptotic optimality in terms of Shannon capacity is proven.

5.5.3 Construction and Decoding of Polar codes

The notion of polarization can be generalized to more than 2×2 precoding, to enlarge the size of the codeblock and make the promise of the polarization theorem come true. As shown in Fig. 5.31, the i-th bitwise polarized channel with input $c[i]$ observes the whole received vector $\mathbf{d}$ *and* the previously decoded i bits $c[0], ..., c[i-1]$. What precoder should we use to get to this situation? Again, when N is sufficiently large a *random polarizer* would do - just a random permutation of the input bits works fine with high probability. This is clearly shown in Fig. 5.32 that displays the capacity of the different bitwise channels in a block of $N = 32$ input bits with a random polarizer on a vector of independent BEC(1/2) (with $\mathcal{C} = 1/2$) channels. The appearance of "extremal" capacity-0 and capacity-1 channels is very clear. Their distribution is roughly 50-50 so that the rate of the code that uses the good channel and freezes the bad channels is $r = 1/2$, just like the capacity of the original scalar channel, and just like the prediction of the polarization theorem.

Unfortunately, a completely unstructured polarizer would not lead to efficient coding and decoding algorithms. But, a nice recursive generalization of the 2×2 encoder of the previous subsection can be devised - although not optimum, it is simple and manageable. To get an order-$N = 4$ precoder (let us call it from now on just *encoder*), we start with the elementary 2×2 encoder shown in Fig. 5.33. The order-2 graph is then doubled vertically, and another layer of $4/2 = 2$ order-2 encoders is appended to the two formers encoders, with the only difference that the inputs of the encoders are now taken with a $4/2$ offset

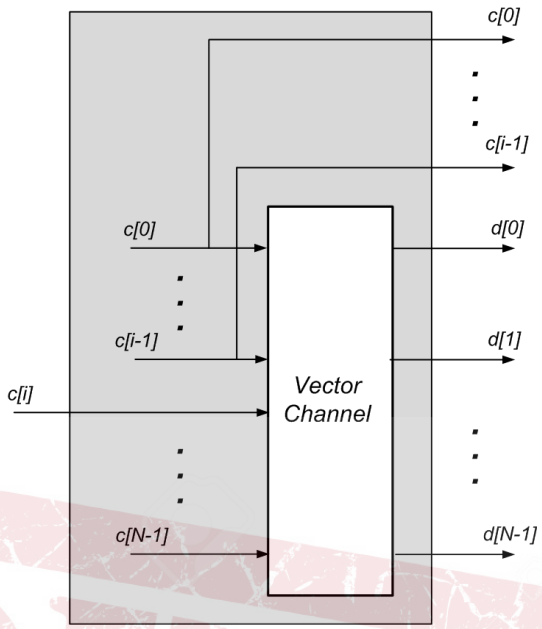


Figure 5.31 Definition of the i-th polarized bitwise channel

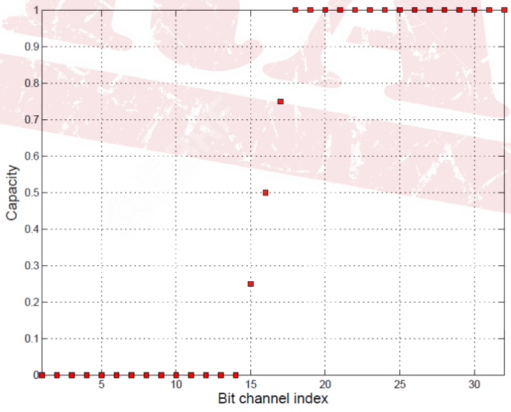


Figure 5.32 Polarization of $N = 32$ BEC(1/2) channels with a random polarizer

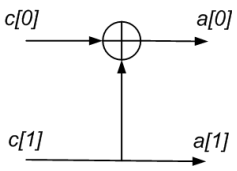
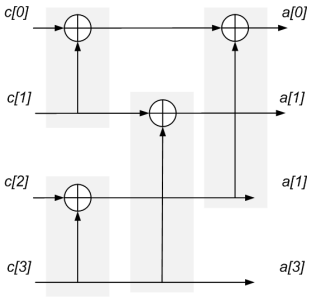


Figure 5.33 Order-2 polar encoder



(b)

Figure 5.34 Recursive construction of order-4 encoder

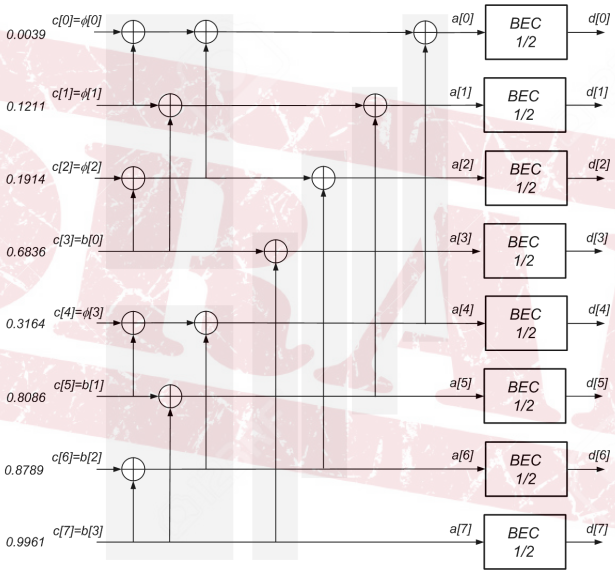


Figure 5.35 Order-8 encoder and channel

between the indices of the inputs, obtaining Fig. 5.34. As the reader can guess, the order-4 scheme can now be extended following the same procedure as before, that is, doubling two order-4 encoder vertically, and appending another layer of 4 4-interleaved $order - 2$ transformations, to give the order-8 encoder in Fig. 5.35. In general, the order-N encoder is thus obtained by stacking two order-$N/2$ encoders and appending another layer of $N/2$ oder-2 encoders with inputs taken at a $N/2$ offset.

This construction can be easily formalized mathematically. The codeword $\mathbf{a}$ that is input to the "parallel channel" is the output of the encoder upon the input $\mathbf{c}$ (containing information and frozen bits) as in Fig. 5.35. The relationship between $\mathbf{c}$ and $\mathbf{a}$ can be generalized for any order $N = 2^m$ and runs as a simple (row-)vector-matrix product $\mathbf{a} = \mathbf{c}\mathbf{G}^{(m)}$, where the order-$N = 2^m$ precoding matrix $\mathbf{G}^{(m)}$ for $m \geq 1$ is constructed

recursively as follows:

$$\mathbf{G}^{(m+1)} = \mathbf{G}^{(m)} \otimes \mathbf{G}^{(1)} \ , \ \mathbf{G}^{(1)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{5.60}$$

where $\otimes$ indicates the (binary) Kronecker product between matrices. Just to make an example,

$$\mathbf{G}^{(2)} = \mathbf{G}^{(1)} \otimes \mathbf{G}^{(1)} = \mathbf{G}^{(1)} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{G}^{(1)} & \mathbf{0}_{2\times 2} \\ \mathbf{G}^{(1)} & \mathbf{G}^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.61}$$

that describes the encoder in Fig. 5.34.

Such power-of-two recursive construction closely resembles the one that is used to obtain the well-known orthogonal binary *Walsh-Hadamard* matrix $\mathbf{H}$ and/or equally orthogonal complex-valued one characteristic of the DFT algorithm $\mathbf{W}$. This suggest that the algorithmic complexity of polar encoding is $N \log N$, just like as fast Fourier or Walsh transform algorithms, therefore well manageable for a real-time digital hardware even for a large blocklength.

5.5.4 Decoding and Practical Examples

Assume now order-8 encoding and an array of 8 BEC(1/2) scalar channels each with capacity $\mathcal{C} = 1/2$ bits/channel use as in Fig. 5.35. We end up with the capacities of the new polarized bitwise channels after encoding that are indicated in the same figure on the left-hand side. A good $r = 1/2$ encoding scheme will therefore dictate to *freeze* the input symbols # 0,1,2, and 4, while using only bits #3,5,6, and 7 to convey information bits. The resulting $(8, 4)$ encoding scheme will be of the form

$$\mathbf{a} = [\phi[0], \phi[1], \phi[2], b[0], \phi[3], b[1], b[2], b[3]]\, \mathbf{G}^{(3)} \tag{5.62}$$

where ϕ indicates a frozen bit, whose actual value is agreed between encoder and decoder (i.e., all 0's), and b is an information bit to be encoded. In general, the split between frozen bits and information bits of $\mathbf{c}$ depends on the properties and status of the channel. In practical codes, a rule is given that is sort of universal, especially for large blocklengths.

The "native" decoding strategy for a polar code follows the definition of the bitwise channel as in Fig. 5.31, and is implemented using the general idea of *successive cancelation* of previously-decoded bits (of course considering the frozen bits as *a-priori* known). In the example in Fig. 5.35, we start by setting $\hat{c}[0] = 0$ since it is a frozen bit, then we go on with $\hat{c}[1] = 0$ and $\hat{c}[2] = 0$ on the same basis. The first real estimation is for $c[3]$, and is based on LAPPRs

$$L(c[3]) = \log \frac{Pr\left\{c[3] = 1 \mid \mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2]\right\}}{Pr\left\{c[3] = 0 \mid \mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2]\right\}} \tag{5.63}$$

or, equivalently (independent equiprobable bits) log-likelihood ratios LLRs:

$$L(c[3]) = \log \frac{Pr\left\{\mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2] \mid c[3] = 1\right\}}{Pr\left\{\mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2] \mid c[3] = 0\right\}} \tag{5.64}$$

From the sign of $L(c[3])$ we derive $\hat{c}[3] = \hat{b}[0]$, that we use to decode $c[5]$, together with $c[4]$ which is frozen:

$$L(c[5]) = \log \frac{Pr\left\{\mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2], \hat{c}[3], \hat{c}[4] \mid c[5] = 1\right\}}{Pr\left\{\mathbf{d}, \hat{c}[0], \hat{c}[1], \hat{c}[2], \hat{c}[3], \hat{c}[4] \mid c[5] = 0\right\}} \tag{5.65}$$

and so on. Generalizing,

$$L(c[n]) = \log \frac{Pr\left\{\mathbf{d}, \hat{c}[0], ..., \hat{c}[n-1] \mid c[n] = 1\right\}}{Pr\left\{\mathbf{d}, \hat{c}[0], ..., \hat{c}[n-1] \mid c[n] = 0\right\}} \tag{5.66}$$

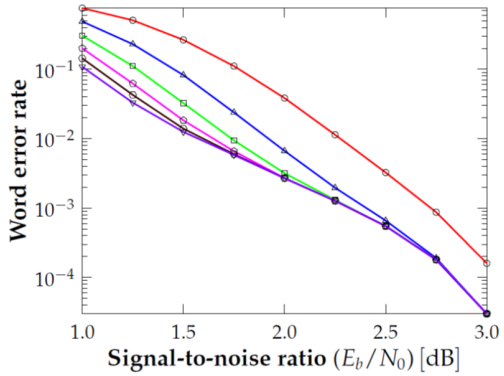
where the previous estimated values are a mixture of frozen and (already detected) information bits. The actual computation of the LLRs depends on the communication channel (BSC, BEC, etc.). For the AWGN, assuming mapping of encoded bits on BPSK/QPSK values, the observed vector is the usual soft-valued vector $\mathbf{r}$ (instead of $\mathbf{d}$) that we have already encountered in our description of Turbo Codes and LDPCs.

The detailed description of the successive-cancelation decoder (SCD) algorithm is outside the scope of this section. The decoding algorithms for Turbo codes and LDPCs are well established, so that we described them in detail in Appendix C; on the contrary, the research on Polar codes is still going on, so that we do not wish to report here something that most likely will be superseded by new, better results in the future. Suffice it to say that the decoder can be cast in a message-passing form that resembles the one used for LDPC decoding. The complexity of the decoder is again $O(N \log N)$ since it follows the power-of-2 recursive structure of the encoder - this is very good, since it is comparable with the linear complexity ($O(N)$) of the LDPC decoder.

Unfortunately, such decoder is not providing optimum performance, so that from this standpoint polar codes are still inferior to LDPC and Turbo. We can mention two main reasons why the SCD performance is not satisfactory: First, once an unfrozen (information) bit is detected in the pertaining decoding step, there is no "going back" and possibly correct a wrong decision in a later iteration (like happens in *real* iterative decoding); second, in the estimation of an information bit $c[i]$, the prior knowledge of frozen bits $c[\ell]$, $\ell > i$ is not considered.

An idea to improve on both aspects is to "defer" decision on information bits as much as possible: this is implemented in the so-called Successive-Cancelation List Decoder (SCLD): instead of doing "pure" SCD, each time a hard decision on an information bit is to be taken, the metrics (LLRs) of both possible values, 0 and 1, are retained and the subsequent decodign step moves from both estimated values. Of course, this would entail an exponential, unmanageable growth of tentative decisions and metrics. The catch is that at each iteration stage only a short list of the L most likely decoding candidates with the largest metrics are retained. This increases the complexity of the decoder to $O(LN \log N)$, just like having L conventional SCDs operating in parallel, but the performance is improved. Figure 5.5.4 compares the BER curves of SCD (red) with those of SCLD for different values of L. Performance is good, but no better than other modern codes'. To date(2020), the SCLD is considered the standard decoding algorithm for polar codes.

The final trick to make polar codes competitive with previous algorithms, especially for short blocks, is the concatenation with an outer algebraic high-rate code, much like what was done for LDPC in DVB-S2. Concatenation here is deeper than in previous instances, since the best decoder of the concatenated code is intrinsically a *joint* decoder that knows of the concatenation. In fact, the CRC is used conventionally as an outer encoder by adding P redundancy bits to the K information bits, before the $N - K - P$ frozen bits are added to



(b)

Figure 5.36 BER Performance of Polar Code with SCD (red) and SCLD (other colors, with different L)

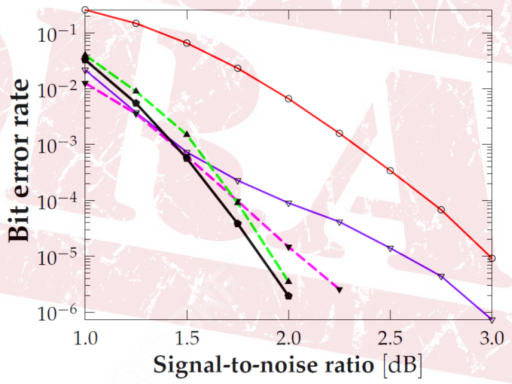


Figure 5.37 BER Performance of Concatenated CRC/Polar Code with SCLD

form the input vector $\mathbf{c}$. Then, during SCLD, out of the list of the L most likely candidates for final decision, only those with a correct CRC are actually considered (*expurgated* list). The relevant performance is shown in Fig. 5.37, where only 16 CRC bits are added to the information bits. This is just the technology adopted (in addition to LDPC) in 5G cellular networks.

5.6 COD/MOD efficiency and the Shannon plane

5.6.1 From AWGN capacity to the efficiency plane

The channel capacity formula allows us to evaluate the performance of the coding and modulation methods (briefly, COD/MOD) on the Gaussian channel.

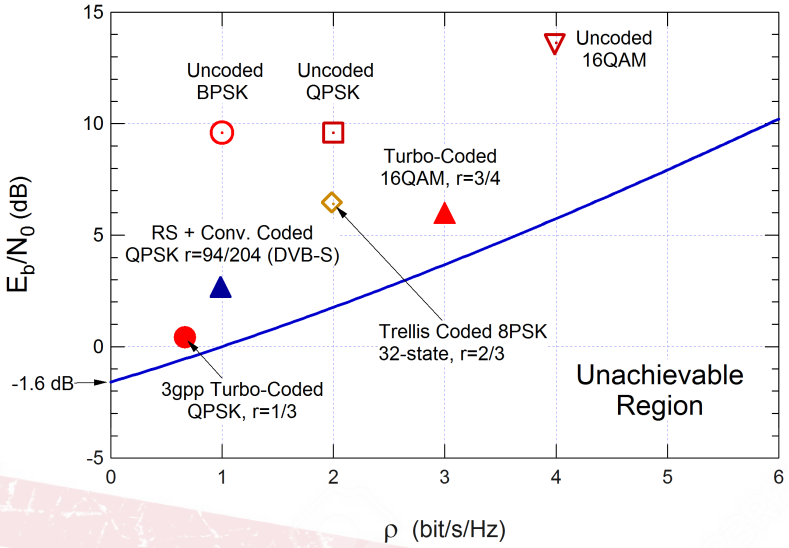


Figure 5.38 Efficiency (Shannon) plane

Reliable communications requires that $R_b \leq \mathcal{C}$ or

$$\frac{R_b}{B} \leq \log\left(1 + \frac{R_b}{B}\frac{P_S T_b}{N_0}\right) \Rightarrow \frac{R_b}{B} \leq \log\left(1 + \frac{R_b}{B}\frac{E_b}{N_0}\right) \tag{5.67}$$

or

$$\frac{E_b}{N_0} \geq \frac{2^{R_b/B} - 1}{R_b/B} \tag{5.68}$$

where E_b is the received energy per bit. We can visualize this relation on the so called *efficiency plane* or *Shannon plane* $(E_b/N_0, R_b/B)$. In particular, the limit curve is

$$\frac{E_b}{N_0} = \frac{2^{R_b/B} - 1}{R_b/B} \tag{5.69}$$

that we see in Fig. 5.38. The region in the figure described by (5.6.1) is "achievable" (with reliable communications), the other is not (see .

The parameter R_b/B is measured in (bit/s)/Hz and denotes the *spectral efficiency* of a COD/MOD scheme. The higher this quantity, the less bandwidth the system requires to convey a given bit rate. A common linguistic error, especially in the Internet world, is to call *bandwidth* the information rate R_b on a link. We know that the former is a physical property of a medium or a signal and is measured in Hertz, whilst the second represents the digital bit rate, is measured in bit/s and depends on the technology with which I exploit the physical bandwidth to implement the digital link, i.e., the COD/MOD method. (channel coding, constellation, and symbol rate). We also see from Fig. 5.38 that if we wish to have a certain spectral efficiency on a link to exploit it at best, I need a minimum quality of the link in terms of signal to noise ratio to achieve reliability. Or, if I revers my point of view, if I have a relatively low SNR on my link, I cannot pretend having a reliable high spectral efficiency.

To make an example, Fig. 5.39 represents a 16QAM constellation with the corresponding *mapping* of the 4 bits associated with each symbol $s_i, i = 1, ..., 16$. Using a channel code,

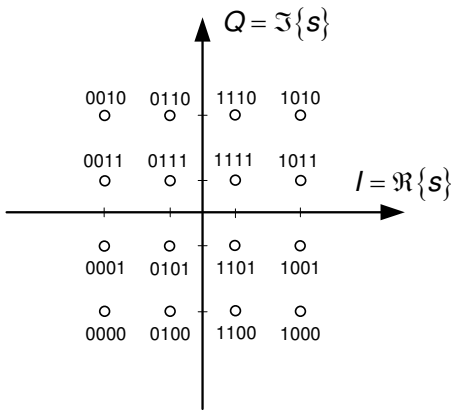


Figure 5.39 16QAM constellation with 4-bit per symbol mapping - Symbol values on both axes are $\pm 1, \pm 3$

the values indicated in the figure are the *binary symbols* $a[n]$ produced by the channel encoder rather than the information bits $b[k]$ (see Fig. 4.12). In general, for a linear I/Q modulation format with M points in the constellation, signaling rate R_s symb/s, and SRRC (Square-Root Raised Cosine) pulse with roll-off factor α, the information bit rate is $R_b = R_s \cdot \log(M) \cdot r$ (r the code rate), and the (RF) bandwidth is $B = (1+\alpha)R_s$. Therefore,

$$\frac{R_b}{B} = \frac{r \cdot \log(M)}{1+\alpha} \tag{5.70}$$

In the optimum case $\alpha = 0$, i.e., with signaling at the Nyquist rate, the spectral efficiency is equal to $rlog(M)$, just equal to the number of (information) bits N_b associated with each constellation symbol.

An interesting limiting case is that of so-called *spread-spectrum* signals, for which the available bandwidth is (on purpose) *much larger* than the bit rate on the link (i.e., $B \gg R_b$): Shannon theorem says that reliable transmission occurs as long as

$$\frac{E_b}{N_0} \geq \left(\frac{E_b}{N_0}\right)_{\min} \triangleq \lim_{R_b/B \to 0} \frac{2^{R_b/B} - 1}{R_b/B} = \ln 2 \cong -1.6\text{dB} \tag{5.71}$$

5.6.2 COD/MODefficiency

Fig. 5.40 shows the characteristic point of a BPSK COD/MOD with the usual $K = 7$ DVB convolutional code with "soft" decoding; note that the point is quite distant from the limiting curve. What is the COD/MOD indicated by the square symbol that comes much closer to this curve? The answer to this question will be given in the next chapter, when we examine the best-performing channel coding schemes that almost reach, as Fig. 5.40 demonstrates. Let's now see how to systematize this type of comparison with any COD/MOD.

To compare the efficiency of a COD/MOD format with channel capacity, we begin by setting a finite, target value of $P(E)$ that is deemed sufficiently small for the application, and then we calculate the value of E_b/N_0 that is required to achieve this performance with the assigned COD/MOD. This nominal value corresponding to the target BER represents the (smaller or larger) *energy efficiency* of the COD/MOD. If we set for instance $P(E) = 10^{-5}$ based on the application requirements, the BER curve on AWGN for uncoded QPSK gives

$E_b/N_0 = 9.6$ dB. This COD/MOD conveys $N_b 2$ bits/symbol (quaternary constellation), therefore the spectral efficiency is R_b/B=2. QPSK modulation without any coding corresponds to the point (2,9.6) on the Shannon plane, as indicated. BPSK modulation will correspond to point (1,9.6) and therefore deviates from the limit curve in the horizontal direction, that is, in the direction of spectral efficiency .

Example 5.32

Let us evaluate the representative points of different COD/MOD used in practice.

i) QPSK with rate-$r = 1/2$ convolutional coding has the same spectral efficiency as uncoded BPSK, i.e., 1 bit/symbol. For the standard DVB convolutional code $r = 1/2$, $K = 7$, the coding gain is 5.2 dB for a $P(E) = 10^{-5}$. This means that the representative point of QPSK encoded with this code is (1,4.4).

ii) For 8PSK modulation with rate-2/3 Ungerböck trellis coding, the spectral efficiency is that of uncoded QPSK, i.e., 2 bits/s/Hz, and the coding gain for the optimal 32-state encoder is about 3.2 dB. The representative point is therefore (2,6.4).

iii) M-QAM modulations increases spectral efficiency, but penalize energy efficiency by about 6 dB for each quadrupling of the constellation points (2 more bits/symbol). In Fig. 5.38 we have indicated the point (4,13.5) representative of 16-QAM.

By appropriately choosing the COD/MOD format, we can "explore" the entire capacity curve by adapting the format to the channel characteristics: higher spectral efficiency if the signal-to-noise ratio is high, lower spectral efficiency if the protection level must be higher because the SNR is poor. This strategy describes the so-called ACM (Adaptive Coding and Modulation) technology typical of modern digital communication standards.

Example 5.33

As for the BSC, to have reliable communications we must ensure that $r \leq c_{BIAWGN}$. Considering the limit case of $r = c$, and recalling that $E_s = rE_b$, we have

$$r + \int_{-\infty}^{\infty} \frac{1}{2}\sqrt{\frac{rE_b/N_0}{\pi}}\left[\exp\left(-r\frac{E_b}{N_0}(c+1)^2\right) + \exp\left(-r\frac{E_b}{N_0}(c-1)^2\right)\right]$$

$$\cdot \log\left(\frac{1}{2}\sqrt{\frac{rE_b/N_0}{\pi}}\left[\exp\left(-r\frac{E_b}{N_0}(c+1)^2\right) + \exp\left(-r\frac{E_b}{N_0}(c-1)^2\right)\right]\right) dc$$

$$+\frac{1}{2}\log\left(\frac{\pi e}{rE_b/N_0}\right) = 0 \tag{5.72}$$

In this limiting case, r coincides with the spectral efficiency: If signaling occurs at a rate of R_s code symbols per second, the source information rate is $R_b = r \cdot R_S$ bits/s. Using the minimum signaling bandwidth, i.e., the Nyquist bandwidth, we also have (for a bandpass-modulated signal) $B = R_s$, therefore $R_b/B = r$.

The relationship that we just obtained, in which the integration must be calculated numerically, implicitly defines the limit curve on the Shannon plane shown in Fig. 5.40 with a thick line, compared with the so-called *unconstrained* capacity curve of the analog Gaussian channel (thin line) already analyzed. From the curve in Fig. 5.40 we can also derive the table of limit values of E_b/N_0 (with $r = c$) that guarantee reliable communication, to be compared with the same values obtainable for low BER from Fig. 4.16 and also reported in the table. The soft-output channel (BIAWGN) is more efficient than the

r	$(\mathbf{E_b/N_0})_{\mathbf{min}}$ BIAWGN (dB)	$(\mathbf{E_b/N_0})_{\mathbf{min}}$ BSC/BPSK (dB)
1/4	-0.80	0.97
1/3	-0.49	1.21
1/2	0.19	1.77
2/3	1.06	2.51

Table 5.2 Limit values of E_b/N_0 for different channels and coding rates

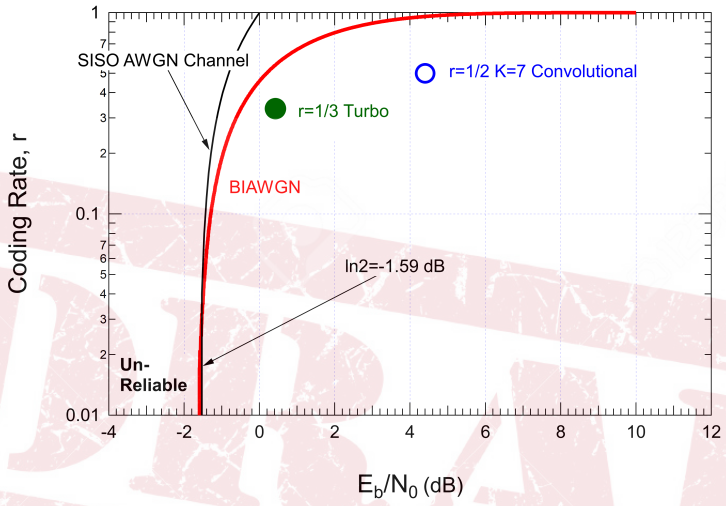


Figure 5.40 Shannon capacity of the BIAWGN channel

hard-detected channel (BSC/BPSK) by more than 1.5 dB.

5.7 Applications of error-correcting codes

We conclude our survey on classical and modern error-correction codes with Table 5.7 showing the *ubiquity* of such technology. In particular, for the major families of codes, we indicate the real-world product or standard technology wherein they are adopted – pretty much any modern digital device!

5.8 Link-Layer Coding with Rateless Codes

In the previous sections and Chapters, we have mentioned several times examples taken from IP communications networks, first and foremost the Binary Erasure Channel in Fig. 4.3 to model end-to-end packet drop events. We have introduced the Polar Codes that were originally devised just for the BEC, and we have also shown that many codes can correct

Code Type	Main Characteristics	Example Real-World Applications / Standards
(A) Block Codes		
Hamming Codes	Simple linear block codes, correct single-bit errors	ECC RAM; Early modems and low-complexity digital links
BCH Codes	Generalization of Hamming codes, strong for random-error correction	Flash memory (NAND); CDs and DVDs; Satellite communication; QR codes (with RS)
Reed–Solomon (RS)	Block code over finite fields, excellent for burst-error correction	CDs, DVDs, Blu-ray discs; QR codes; DSL/cable modems; NASA deep-space (CCSDS); DVB broadcasting
Reed–Muller (RM)	Structured linear block code, useful in moderate-noise channels	Deep-space and satellite communication; Space probe telemetry
Low-Density Parity-Check (LDPC)	Sparse matrix block code, near-Shannon-limit performance	Wi-Fi (IEEE 802.11n/ac/ax); 5G NR (data channels); DVB-S2/S2X; 10GBASE-T Ethernet; Hard drives (data storage)
Polar Codes	Capacity-achieving block codes with efficient decoding	5G NR (control channels); Optical and satellite systems
Product / Concatenated Codes	Combination of multiple block codes for improved protection	Space communication (RS + convolutional); Magnetic tape and optical media
(B) Convolutional and Iterative Codes		
Convolutional Codes	Continuous-stream encoding with memory; decoded via Viterbi algorithm	GSM and 3G systems; Wi-Fi (802.11a/b/g); Satellite links; NASA telemetry
Turbo Codes	Parallel concatenated convolutional codes with iterative decoding	3G (UMTS), 4G LTE; Deep-space communication (CCSDS); DVB-RCS; CDMA2000

Table 5.3 Common error-correcting code families and their real-world applications.

erasures: take the example of decoding a punctured codes as in Fig. 5.19, where we are forced to intentionally introduce erasures in the stream that is presented to the decoder - recovering from an erasure is no different than revealing and correcting an error. But, adding a-priori a large amount of redundancy to recover form erasure (and we're still thinking of lost packets) may not be the best end-to-end strategy when we have available some form of *feedback information*. As is known, when the destination detects an erasure event (e.g., a wrong packet CRC), may wish to send a NACK (Negative Acknowledgement of Recepit) message back to the sender, and request the lost piece of information again. The repeat request is typically carried out in the OSI stack at the *Data Link* layer, once the physical layer has done its best (including channel coding) to deliver delivered correct bits. If the packet is lost it is the (Data) Link Layer that manages this higher-layer event.

Repeating the lost packet (even more than once if the drop probability is high) can be considered as a further layer of redundancy added to the information packet - another layer of coding that is carried out at the Link Layer, and that we call in general *Link-Layer Coding*.

We already know that repetition is the most naive form of coding, therefore we may wonder wether some more advanced and more efficient form of LLC exists: the answer lies in one seminal paper by M. Luby that in 2002 introduced the notion of Fountain Codes.

5.8.1 Rateless/Fountain Codes

The main idea behind Fountain Codes (FC) is fantastically simple: assume that the source has to send out a message (a file of music for instance) made of K packet of fixed length ℓ, for a total size of $K\ell$ bits. The ideal Fountain encoder is a (thoretically endless) source (fountain) of length-ℓ encoded blocks of bits obtained form the original messagge, with the property that as soon as the destination receives K' of these encoded packets, with K' slightly larger that K, then the original message can be recovered, regardless of the order with which such packets are received. This means that if in the course of the session some packet is lost, the message is recovered nonetheless as soon as the first available K' packets are received - the context is that of a *block* erasure channel.

The definition also justifies why Fountain Codes are labeled *rateless*: their actual coding rate (redundancy) is undefined as it is case-dependent - if the channel is good the message is soon retrieved and the rate is high, is the channel is bad we need more coded packets to be sent since many will be lost, and the rate is lower.

Fountain codes are pretty good both for one-to-one communication with feedback and for multicast/broadcast: in the first case, as soon as the message is retrieved a positive ack message is sent back to the sender and the fountain stopped; in the second case, a maximum run of the fountain is allowed, hoping that all end-user will need no more packets than the maximum - anyway, as soon as his/her adequate numer of packets is transmitted, any individual user will decode the whole message well before the end of the run.

The same technology is being used for distributed storage of large chunks of data (our $K\ell$ message) on different media (think of a RAID hard-disk storage system). If the chunk (file) is large, it is very likely that some of the block of the stored file will be corrupted during write or read from different physical media (different hard disks). So, instead of writing the original message we write a Fountain of message blocks (packets) to be robust against storage failures.

A final application area of FCs is the distributed download of a large file (a movie, a SW update etc.) from multiple servers. The servers may run independently multiple fountains with the same message, and the client will get the whole information as soon as it receives the sufficient number of packets, independent of the actual server(s) from which they were received, speeding up very much the download with no need of coordination across servers.

5.8.2 The Random Linear FC

Assume that we have a message made of K ℓ-bit-long packets $\mathbf{b}[0], \mathbf{b}[0][1], \ldots, \mathbf{b}[K-1]$ (think of $\mathbf{b}[0]$ as a row binary vector containing the packet bits). The message is first buffered (so that all of the packets are available for coding), and then the Fountain starts computing ℓ-bit-long coded packets $\mathbf{a}[i]$, $i = 0, 1, \ldots$ as alinear combinations of all of the packets in the message:

$$\mathbf{a}[i] = \bigoplus_{k=0}^{K-1} g_{i,k}\mathbf{b}[k] = \tag{5.73}$$

The coefficients $g_{i,k}$ of the (binary) linear combination are *random independent bits* generated by the fountain each time a new encoded packet $\mathbf{a}$ has to be generated (random coding

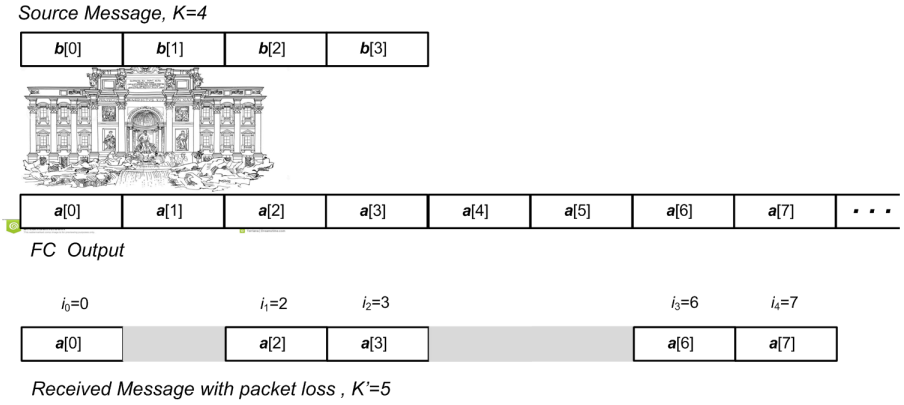


Figure 5.41 Fountain Code with packet loss

again!). The encoding coefficients will not be of course totally random - they will be a PN sequence known to the encoder and the user decoder. Now, the destination will get the first K' packets $\mathbf{a}[i_0], \mathbf{a}[i_1], \ldots, \mathbf{a}[i_{(K'-1)}]$ that are not erased and will try to decode the original message[4], as sketched in Fig. 5.41. Since che encoding coefficients are random, it is pretty apparent that any packet is equivalent to any other in the decoding procedure. If we collect the K' packets in a row we obtain a $1 \times K'$ array of packets that can be related to the $1 \times$ K row array of source packets as follows:

$$[\mathbf{a}[0], \mathbf{a}[1], \ldots, \mathbf{a}[K'-1]] = [\mathbf{b}[0], \mathbf{b}[1], \ldots, \mathbf{b}[K-1]]\,\mathbf{G} \tag{5.74}$$

or, for short, $\mathbf{A} = \mathbf{BG}$, where $\mathbf{G} = \{g_{i,k}\}$ is the $K \times K'$ *generator matrix* of this random code. Since $\mathbf{G}$ is random, we are not 100% certain to be able to recover the source bits. This only happens if $\mathbf{G}$ is full-rank, i.e, we can find in $\mathbf{G}$ K linearly independent column, and construct with those an invertible submatrix $\mathcal{G}$ so that $\mathbf{B} = \mathbf{A}\mathcal{G}^{-1}$. This only happens with a certain probability, let's say $1 - p_{DF}$, and if does not happen we have a decoding failure - with probability p_{DF}. The fundamental result of Random Transform Coding is that,

$$p_{DF} \leq 2^{-(K'-K)} \tag{5.75}$$

and the bound is very tight for large K. If we wish to reduce the packet loss rate after decoding below a certain p_{DF}, we will incur into an average *overhead* (additional coded packets to be sent) equal to

$$K' - K \geq \log_2\left(\frac{1}{p_{DF}}\right) \tag{5.76}$$

so that the (average) coding rate at the destination is

$$r \leq \frac{1}{1 - \log_2(p_{DF})/K} \tag{5.77}$$

and the (average) coding rate at the sender, considering a packet drop probability p_{CE} is

$$r \leq \frac{1 - p_{CE}}{1 - \log_2(p_{DF})/K} \tag{5.78}$$

[4]We don't know how many fountain packets have been sent out before we get these K' because there can be lost packets in between; we also don't exactly know the values of the index i of the received packets for the same reason: this is why we have used the complicated notation $i_{k'}, k' = 0, 1, \ldots, K'-1$

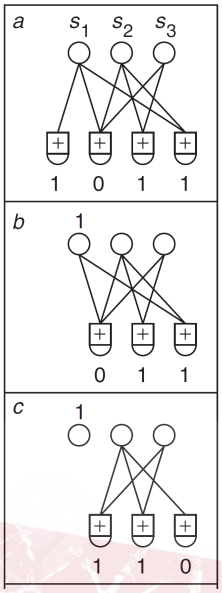


Figure 5.42 Decoding of the Luby-transform FC (from [10])

The good news with random transform coding is that p_{DF} can be made arbitrarily small by increasing $K' - K$; the bad news is that the complexity of coding is quadratic (matrix multiplication) and that of decoding is cubic (matrix inversion) with K - too large to be used in practice.

5.8.3 The Luby Transform FC

How to retain the good performance of the RL code while decreasing complexity? We still produce coded packets of size ℓ as in the RL code: This time, instead of using random equiprobable coefficients, we set their value according to a different probability distribution, so that they became *sparse*, i.e., $\mathbf{G}$ becomes low-density! In particular, to compose packet $\mathbf{a}[i]$, we set first the number d of 1's in the set of the $\{g_{i,k}\}_{k=0}^{K-1}$, so that $\mathbf{a}[i]$ will be the sum of d packets - which packets to choose doesn't matter, we will just take D packets $\mathbf{b}[k]$ at random. Of course, the probability distribution $p(d) \stackrel{\triangle}{=} \Pr\{D = d\}$ of the random number D (the so-called *degree* of the coded packet $\mathbf{a}[i]$), will be such that "small" values $d \ll K$ will be highly probable, so that the set of coefficients $\{g_{i,k}\}_{k=0}^{K-1}$ (and so the generation matrix $\mathbf{G}$) will be low-density. The complexity of encoding is now reduced to $K'\mathrm{E}\{d\}$ that is almost linear in K if $\mathrm{E}\{D\} \ll K$.

Decoding proceeds by successive cancelations, and can be easily visualized. Let us associate to $\mathbf{G}$ a (low-density) Tanner graph, where the bitnodes represent the K packets $\mathbf{b}[k]$ to be decoded, and the checknodes are the K' encoded packets $\mathbf{a}[i]$ - we show an example in Fig. 5.42 where packets are just made by $\ell = 1$ bit, $K = 3$, $K' = 4$ and the received packets $a[i]$ are 1011. First, to be able to decode from those received K' packets, we have to make sure that no bitnode is "isolated", i.e., in our random choice of source packets building the K' coded packets, we have to make sure that *all* source packets have been used, otherwise we have a decoding failure. This is a constraint to be well kept in mind

when designing the degree probability distribution $p(d)$, in addition to the desired average "density" of edges of the Tanner graph. Second, decoding can only start if (at least) one of the $K' \geq K$ received packets has degree 1 as in Fig. 5.42 (a)(remember that the encoding coefficients are known to the decoder) - if there's no such packet, either decoding fails if we have a time constraint, or we have to wait for more packets from the fountain. Let us call this initial packet $\mathbf{a}[i_0]$. The checknode does the decoding $\mathbf{b}[i_0] = \mathbf{a}[i_0]$ and sends $\mathbf{b}[i_0]$ up to the (only) bitnode to which it's attached (Fig. 5.42 (b)). The bitnode stores this value and sends it down to all of the other checknodes to which it's attached. Each checknode adds this message to its current stored value ("canceling" the decoded packet), the bitnode $\mathbf{b}[i_0]$ and all of the edges attached to it are removed form the graph, obtaining Fig. 5.42 (c), and another decoding iteration just like the one we have described is started on the reduced graph: we go and search for a degree-1 checknode and we decode $\mathbf{b}[i_1]$... until all packets are decoded.

Clearly, decoding failure can happen at any decoding iteration, so that careful design of the weight distribution is required: on one hand we wish to have low degrees to keep encoding low-density ad have degree-1 code packets; on the other, we occasionally need high-degree coded packets to make sure that we have no "isolated" source packets. Luby found that the minimum average degree to "cover" all of the source packets without leaving anyone behind is $\mathrm{E}\{D\} = \ln(K)$. This minimum number also guarantee the minimum "density" of the Tanner graph and therefore the minimum decoding complexity $\mathrm{E}\{D\} \cdot K = K \ln(K)$. A degree distribution $p(d)$ that has (roughly) this average value is the so-called *soliton* distribution:

$$p_S(d) = \begin{cases} \frac{1}{K} & d = 1 \\ \frac{1}{d(d-1)} & 2 \leq d \leq K \end{cases} \tag{5.79}$$

Although degree-one packets are highly probable, it is found that the relatively high probability of the other values causes frequent decoding failures. A better distribution is a modification of the soliton distribution, called *robust soliton*, as follows:

$$p_{RS}(d) = \frac{p_S(d) + p_\tau(d)}{P} \tag{5.80}$$

where $P = 1 + \sum_{d=1}^{K} p_\tau(d)$ is just a normalization constant, and where the additional distribution $p_\tau(d)$ depends on two parameters as follows:

$$p_\tau(d) = \begin{cases} \frac{\Delta}{K} \cdot \frac{1}{d} & d = 1, 2, \ldots, K/\Delta - 1 \\ \frac{\Delta}{K} \cdot \ln \frac{\Delta}{\delta} & d = K/\Delta \\ 0 & d = K/\Delta + 1, \ldots, K \end{cases} \tag{5.81}$$

The new component in the degree distribution introduces a spike at the (relatively large) value $d = K/\Delta$ (where Δ is a design parameter) to ensure that packets with that (relatively large) degree are highly probable, and no "isolated" packet remains before decoding. The second parameter δ is related to the p_{DF}: Luby managed to show that following the robust soliton distribution (5.81), the average number of packets to be collected in order for $p_{DF} < \delta$ is $K' = P \cdot K$.

Example 5.34

Imagine we need to save a 100 MB-file segmented into $K = 100,000$ blocks (packets) with length $\ell = 1024 \times 8$ bits each.

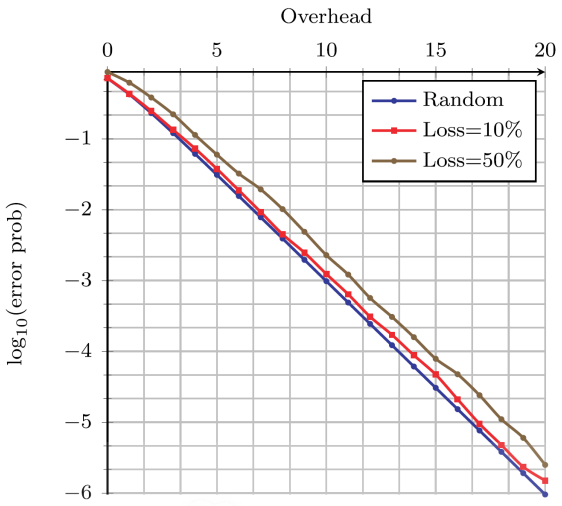


Figure 5.43 Performance of the standard R10 Raptor code (from [12])

5.8.4 Raptor Codes

Optimizing the degree distribution led to a manageable $K\ln(K)$ complexity for the LTFC. Can we do better? The $\ln(K)$ factor in the complexity comes from the average degree introduced by the optimized robust soliton distribution to guarantee that packets are not "isolated" and decoding does not fail (to a large probability $1-p_{DF}$). Shokrollahi had the idea to use an " imperfect" LTFC with average degree close to 3, therefore linear complexity $3K$, intentionally allowing a *controlled* number of decoding failures after reception of K' coded packets. This is tantamount to allowing a controlled number of *erasures* in the end-to-end channel after fountain decoding, erasures that can be eliminated by an outer concatenated code at the packet level - we can call them *outer* erasures at the output of the FC decoder (caused by decoding failures), not to be confused with the *inner* erasures that we have already considered, introduced by transmission of the FC-encoded packets. Shokrollahi proposed for the purpose an irregular LDPC code for which, like in the LT decoder, bitnodes and checkodes operate on packets rather than bits, and called this arrangement "Rapid Tornado (RapTor)" code.

The design and optimization of Raptor codes is complicated, but they are very effective. They have been standardized for 3G and 4G multimedia multicast over wireless networks. A sample performance of the so-called R10 code is shown in Fig. 5.43 in terms of failure probability p_{DF} as a function of the code overhead $(1-p_{CE})(K'-K)/K\%$ for $K=1000$. The performance is compared with the theoretical one of the linear random FC showing that it is very close even with relatively high packet loss rate p_{CE}.

APPENDIX A

THE BCJR ALGORITHM

We intend to develop an algorithm to compute the LAPPRs

$$L(b[k]) \triangleq \ln \frac{\Pr\{b[k]=1|\mathbf{r}\}}{\Pr\{b[k]=0|\mathbf{r}\}} \tag{A.1}$$

for a rate-1/2, possibly recursive-systematic, convolutionally encoded block of K information bits $\mathbf{b} = [b[0], b[1], ..., b[K-1]]$, assuming that $\mathbf{r}$ is the noise-corrupted output of an AWGN channel. Using Bayes' rule and considering that the information bits are equiprobable

$$\Pr\{b[k]=1|\mathbf{r}\} = f(b[k]=1, \mathbf{r}) / f(\mathbf{r}) \tag{A.2}$$

where $f(\cdot)$ indicates a (joint or marginal) probability density function (pdf), and where we omit the subscript $_{\mathbf{R}}$ for simplicity. Using the same rule for the denominator of (A.1), we get

$$L(b[k]) = \ln \frac{f(b[k]=1, \mathbf{r})}{f(b[k]=0, \mathbf{r})} \tag{A.3}$$

Consider now that the *event* $\{b[k]=1(0)\}$ can be partitioned into the union of all of the disjoint events given by all those transitions in the code trellis that brings from a certain state i at time $k-1$ to another state j at times k and that are "labeled" by $b[k]=1(0)$. Based on this, our LAPPR becomes:

$$L(b[k]) = \ln \frac{\sum_{(i,j):b[k]=1} f(\xi[k-1]=i, \xi[k]=j, \mathbf{r})}{\sum_{(i,j):b[k]=0} f(\xi[k-1]=i, \xi[k]=j, \mathbf{r})} \tag{A.4}$$

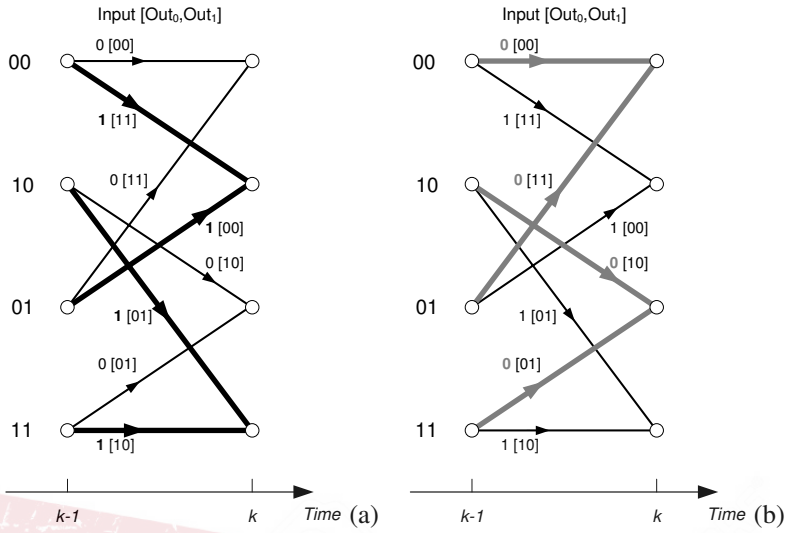


Figure A.1 State transitions in a trellis corresponding to $b[k] = 1$ (a) and $b[k] = 0$ (b)

where $\xi[k]$ represents the state of the decoder at time k. If we take the usual convolutional code of Fig. 5.5 as our paradigmatic example, the set involved in the sum at the numerator of (A.4) is shown highlighted in Fig. A.1 (a), and that involved in the sum at the denominator is shown in Fig. A.1 (b). The key result is showing that the two joint pdf's above (that we will denote for short as $f\left(i,j,\mathbf{r}\right)$) can be computed by the two forward-backward recursion we have mentioned in the main text. First, we use the following self-evident notation (recall that r=1/2):

$$\mathbf{r} = \left[\overbrace{r[0], r[1], ..., r[2k-2], r[2k-1]}^{\mathbf{r}^-}, \overbrace{r[2k], r[2k+1],}^{\mathbf{r}_k}\right.$$
$$\left.\overbrace{r[2k+2], r[2k+3], ..., r[2N-2], r[2N-1]}^{\mathbf{r}^+}\right] \tag{A.5}$$

Then, we observe that

$$\begin{aligned} f\left(i,j,\mathbf{r}\right) &= f\left(i,j,\mathbf{r}^-,\mathbf{r}_k,\mathbf{r}^+\right) \\ &= f\left(\mathbf{r}^+|i,j,\mathbf{r}^-,\mathbf{r}_k\right) f\left(i,j,\mathbf{r}^-,\mathbf{r}_k\right) \\ &= f\left(\mathbf{r}^+|i,j,\mathbf{r}^-,\mathbf{r}_k\right) f\left(j,\mathbf{r}_k|i,\mathbf{r}^-\right) f(i,\mathbf{r}^-) \end{aligned} \tag{A.6}$$

In the first term of the last equation above, once the state at the time k is known to be equal to j, $\mathbf{r}^+$ becomes (conditionally) independent of everything that has happened until time k. Similarly, in the second term, once state i at time $k-1$ is assigned, the past $\mathbf{r}^-$ is irrelevant. The conclusion is that

$$\begin{aligned} f\left(i,j,\mathbf{r}\right) &= f\left(\mathbf{r}^+|j\right) f\left(j,\mathbf{r}_k|i\right) f(i,\mathbf{r}^-) \\ &\triangleq \beta^{(j)}[k+1] \cdot \Gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1] \end{aligned} \tag{A.7}$$

and so

$$L(b[k]) = \ln \frac{\sum_{(i,j):b[k]=1} \beta^{(j)}[k+1] \cdot \Gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1]}{\sum_{(i,j):b[k]=0} \beta^{(j)}[k+1] \cdot \Gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1]} \tag{A.8}$$

Now, the final and fundamental result is the recursive rule for the computation of $\alpha^{(i)}[k-1]$ and $\beta^{(j)}[k+1]$ above. First, we review how to compute $\Gamma^{(j,i)}[k]$:

$$\begin{aligned} \Gamma^{(j,i)}[k] &= f\left(j, \mathbf{r}_k | i\right) = f\left(i, j, \mathbf{r}_k\right) / p(i) \\ &= \frac{f\left(i, j, \mathbf{r}_k\right)}{p(i,j)} \frac{p(i,j)}{p(i)} \\ &= f\left(\mathbf{r}_k | i, j\right) p(j|i) \end{aligned} \tag{A.9}$$

Conditioning on i and j in the first term of the last equation above means knowing in advance the value of $a_0[k]$ and $a_1[k]$, so that $f\left(\mathbf{r}_k | i, j\right) = f\left(\mathbf{r}_k | a_0[k], a_1[k]\right)$; in addition, $p(j|i)$ is the probability of a given state transition. This probability is either equal to 0 if the transition is not in the trellis of the code we are considering, or it is equal to the probability of $b[k]$ being equal to 1 or to 0 for the allowed transitions. The conclusion is that, on the trellis,

$$\begin{aligned} \Gamma^{(j,i)}[k] &= \frac{1}{2} f\left(\mathbf{r}_k | a_0[k], a_1[k]\right) \\ &= \frac{1}{4\pi\sigma^2} \exp\left[-\frac{(r[2k] - a_0[k])^2 + (r[2k+1] - a_0[k])^2}{2\sigma^2}\right] \end{aligned} \tag{A.10}$$

where $\sigma^2 = (2E_s/N_0)^{-1}$ is the variance of the AWGN component affecting the coded symbols. In the following, we will use instead of this a *normalized*, version of Γ that leads to the same results and is easier to compute. [1]

$$\gamma^{(j,i)}[k] \triangleq \exp\left[-\frac{(r[2k] - a_1[k])^2 + (r[2k+1] - a_2[k])^2}{2\sigma^2}\right] \tag{A.11}$$

We see that γ is akin to the branch metrics of the VD, and can be easily computed on the trellis based upon the observed signal samples $\mathbf{r}$ and with the knowledge of the noise variance (that, by the way, is not required by MLSE implemented through a VD).

Let us now go back to the computation of $\alpha^{(i)}[k-1]$. If we consider one more step, and go to state j at time k, we have:

$$\begin{aligned} \alpha^{(j)}[k] &= f(j, \mathbf{r}^-, \mathbf{r}_k) \\ &= \sum_i f(i, j, \mathbf{r}^-, \mathbf{r}_k) \\ &= \sum_i f(j, \mathbf{r}_k | i, \mathbf{r}^-) f(i, \mathbf{r}^-) \\ &= \sum_i \Gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1] \end{aligned} \tag{A.12}$$

This is a recursive equation similar to the one for the accumulated metric of the VD (5.41). From now on, we will actually use the normalized version

$$\alpha^{(j)}[k] = \sum_i \gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1] \tag{A.13}$$

[1] Notice that any normalization constant vanishes in the ratio in (A.8)

Again, the $\gamma^{(j,i)}[k]$ play the role of branch metrics, and the α's we get at times k are one for each allowed state in the trellis, as in the VD.

A similar but subtler trick is required to derive a relation to compute $\beta^{(j)}[k+1]$. If we now go *backwards* in time by one step, we have

$$\begin{aligned}\beta^{(i)}[k] &= f\left(\mathbf{r}_k, \mathbf{r}^+|i\right) \\ &= \sum_j f\left(\mathbf{r}_k, \mathbf{r}^+, j|i\right) \\ &= \sum_j f\left(\mathbf{r}^+|\mathbf{r}_k, i, j\right) f(j, \mathbf{r}_k|i) \\ &= \sum_j f\left(\mathbf{r}^+|j\right) \cdot \Gamma^{(j,i)}[k] \end{aligned} \tag{A.14}$$

where we considered that once state j at time k is assigned, there is no dependence left on $\mathbf{r}_k, i$. Normalizing,

$$\beta^{(i)}[k] = \sum_j \beta^{(j)}[k+1] \cdot \gamma^{(j,i)}[k] \tag{A.15}$$

This is a *time-reversed* recursive equation with another kind of accumulated metrics and with (again) branch metrics $\gamma^{(j,i)}[k]$. This explains why we said in the main text that the BCJR needs two, intertwined, forward-backward VD-like detectors to compute the LAPPRs. Once γ, α, and β are computed for every k in the block of N information bits according to (A.10), (A.13), and (A.15), respectively, the LAPPRs are easily derived through (A.8). But this, contrary to the VD, calls for *non-causal* processing with a decoding delay equal to the whole length of the block N.

It turns out that the implementation of the BCJR requires some care, as the algorithm as it stands is numerically ill-conditioned. To make it better, we have to replace the probabilities that we have computed until now with *log*-probabilities to make it numerically stable. In addition, the application of the algorithm to turbo decoders needs to devise a modified SISO detector based on the BCJR, but with an *additional input* that represents the *input* extrinsic information received from the twin SISO detector in the decoder, as well as an *additional output* that represents the *output* extrinsic information to be handed over back to the twin detector again. Extrinsic means *external*, something that the detector cannot compute itself.

To do so, we resort to a description of the BCJR in the *log*-probability domain. If we use the shorthand notation $\overline{\gamma}$ for $\ln \gamma$, $\overline{\alpha}$ for $\ln \alpha$, $\overline{\beta}$ for $\ln \beta$, and $L[k]$ for $L\,(b[k])$, it is clear that

$$\begin{aligned} L[k] = \ln \sum_{(i,j):b[k]=1} \exp\left(\overline{\beta}^{(j)}[k+1] + \overline{\gamma}^{(j,i)}[k] + \overline{\alpha}^{(i)}[k-1]\right) \\ - \ln \sum_{(i,j):b[k]=0} \exp\left(\overline{\beta}^{(j)}[k+1] + \overline{\gamma}^{(j,i)}[k] + \overline{\alpha}^{(i)}[k-1]\right) \end{aligned} \tag{A.16}$$

We focus now on how to inject extrinsic information into the BCJR algorithm in the form of an *a-priori* LAPPR (that for us will come from the twin detector in the turbo decoder). If we use the shorthand notation $\mathbf{a}[k] = [a[2k], a[2+k1]]$ and $\mathbf{r}[k] = [r[2k], r[2+k1]]$,

then[2]

$$\begin{aligned}\gamma^{(j,i)}[k] &= p\left(b[k]\right)\cdot f\left(\mathbf{r}_k|\mathbf{a}[k]\right) \\ &= p\left(b[k]\right)\exp\left[-\frac{\|\mathbf{r}[k]-\mathbf{a}[k]\|^2}{2\sigma^2}\right]\end{aligned} \tag{A.17}$$

If we want to keep into account the possible "unbalance" of the (prior) probabilities of 1 and 0, we cannot just say as in the derivation of the standard BCJR above that $p\left(b[k]\right)$ is equal to 1/2, but we have to accept any soft value for it. These "unbalanced" probabilities are just the extrinsic information we are seeking for, and we have to accept that this piece of extrinsic information, in the form of a log-probability ratio, comes form outside of the detector. So, our extrinsic LAPPR is

$$L_e[k] \triangleq \ln\frac{\Pr\{b[k]=1\}}{\Pr\{b[k]=0\}} = \ln\frac{p_1}{p_0} = \ln\frac{p_1}{1-p_1} = \ln\frac{1-p_0}{p_0} \tag{A.18}$$

so that, after some math, and further simplifying notation,

$$p_1 = \frac{\exp(L^e/2)}{2\cosh(L^e/2)} \quad , \quad p_0 = \frac{\exp(-L^e/2)}{2\cosh(L^e/2)} \tag{A.19}$$

The two values in (A.19) can be summarized as follows:

$$p\left(b[k]\right) = \frac{\exp(b_\pm[k]L_e[k]/2)}{2\cosh(L^e/2)} = A(L_e[k])\exp\left(\frac{b_\pm[k]\cdot L_e[k]}{2}\right) \tag{A.20}$$

where we introduced the antipodal value of $b[k]$, namely, $b_\pm[k]=2b[k]-1$. Now we know how to embed into the BCJR algorithm the extrinsic information - we only need to properly modify the computation of the branch metrics γ on the trellis as follows:

$$\gamma^{(j,i)}[k] = A(L_e[k])\exp\left(\frac{b_\pm[k]\cdot L_e[k]}{2}\right)\exp\left[-\frac{\|\mathbf{r}[k]-\mathbf{a}[k]\|^2}{2\sigma^2}\right] \tag{A.21}$$

so that

$$\overline{\gamma}^{(j,i)}[k] = \ln\left[A(L_e[k])\right] + \frac{b_\pm[k]\cdot L_e[k]}{2} - \frac{\|\mathbf{r}[k]-\mathbf{a}[k]\|^2}{2\sigma^2} \tag{A.22}$$

We have now to reformulate the recursions on $\overline{\alpha}$ and $\overline{\gamma}$ in the log domain, to obtain the value of $L(b[k])$ that keeps into consideration the input extrinsic information, *and* we also have to enucleate within that value the *incremental* contribution that has actually come from the computation of the BCJR and has not come from the input $L_e[k]$. This will become the "new" *output* $L_e[k]$ to be handed over back to the other twin detector.

If we go back to (A.16), we don't see any simplification with respect to its non-log counterpart (A.8), especially because we have to compute generic terms of the kind of $\ln\left[\exp(a)+\exp(b)\right]$. The funny thing is that, on the contrary, the simplification we seek for lies just here. In fact, the reader can easily verify that

$$\ln\left[\exp(a)+\exp(b)\right] = \max(a,b) + \ln\left[1+\exp(-|a-b|)\right] \tag{A.23}$$

[2]Throughout this Appendix, we will always intend that the code symbols $a[n]$ are antipodal, i.e., they belong to the alphabet $\{\pm 1\}$.

This nonlinear two-operand function is the starting point to define the following *operator*:

$$a \max^* b \stackrel{\triangle}{=} \ln\left[\exp(a)+\exp(b)\right] = \max(a,b) + \ln\left[1+e^{-|a-b|}\right] \tag{A.24}$$

The operator can be generalized to more than two operands, and can be shown to be associative (trivial):

$$\begin{aligned} a \max^* b \max^* c &= (a \max^* b) \max^* c = a \max^* (b \max^* c) \\ &= \ln\left[\exp(a)+\exp(b)+\exp(c)\right] \end{aligned} \tag{A.25}$$

It also obeys a nice distributive law with respect to the sum (proof left for exercise):

$$(z+a) \max^* (z+b) = z + (a \max^* b) \tag{A.26}$$

We will use in the following the compact notation

$$\begin{aligned} \max_{i=1}^{N}{}^*(a_i) &\stackrel{\triangle}{=} a_1 \max^* a_2 \max^* \ldots \max^* a_N \\ &= \ln\left[\exp(a_1)+\exp(a_2)+\ldots+\exp(a_N)\right] \end{aligned} \tag{A.27}$$

Now, equation (A.13) says

$$\alpha^{(j)}[k] = \sum_i \gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1]$$

so that

$$\begin{aligned} \overline{\alpha}^{(j)}[k] &= \ln\left(\sum_i \gamma^{(j,i)}[k] \cdot \alpha^{(i)}[k-1]\right) \\ &= \max_i{}^*\left(\overline{\gamma}^{(j,i)}[k] + \overline{\alpha}^{(i)}[k-1]\right) \end{aligned} \tag{A.28}$$

and similarly, from (A.15),

$$\overline{\beta}^{(i)}[k] = \max_j{}^*\left(\overline{\beta}^{(j)}[k+1] + \overline{\gamma}^{(j,i)}[k]\right) \tag{A.29}$$

so that (A.16) reads

$$\begin{aligned} L[k] = &\max_{b[k]=1}{}^*\left(\overline{\beta}^{(j)}[k+1] + \overline{\gamma}^{(j,i)}[k] + \overline{\alpha}^{(i)}[k-1]\right) \\ &- \max_{b[k]=0}{}^*\left(\overline{\beta}^{(j)}[k+1] + \overline{\gamma}^{(j,i)}[k] + \overline{\alpha}^{(i)}[k-1]\right) \end{aligned} \tag{A.30}$$

After this considerable exercise of algebra, we have to identify the contribution in $L[k]$ as found in (A.30) above that can be considered *extrinsic* to the twin detector, as stated before. This requires again considerable computations. We reconsider (A.22) and we use a slightly modified expression for $\overline{\gamma}^{(j,i)}[k]$:

$$\overline{\gamma}^{(j,i)}[k] = \ln\left[A(L_e[k])\right] + \frac{b_{\pm}[k] \cdot L_e[k]}{2} - \frac{\|\mathbf{r}[k]\|^2 + \|\mathbf{a}[k]\|^2 - 2\mathbf{r}[k] \cdot \mathbf{a}[k]}{2\sigma^2} \tag{A.31}$$

Observe that w When we use (A.31) in (A.30), all terms that do *not* depend on the particular transition (i, j) vanish in the computation of the difference between the two $\max^*$ functions. This is the case for the term $\ln[A(L_e[k])]$, as well as for the terms $\|\mathbf{r}[k]\|^2/2\sigma^2$ and $\|\mathbf{a}[k]\|^2/2\sigma^2$. The terms left to be used in (A.30) are thus $b_\pm[k] \cdot L_e[k]/2$ and $r[2k]a_1[k]/\sigma^2 + r[2k+1]a_2[k]/\sigma^2$.

Recall finally that for a rate-1/2 systematic code, $a_1[k] = b_\pm[k]$, and $a_2[k] = p_\pm[k]$ (antipodal symbol), so that

$$
\begin{aligned}
L[k] = \max_{b_\pm[k]=1}{}^* \Bigg(& \overline{\beta}^{(j)}[k+1] + \frac{L_e[k] \cdot b_\pm[k]}{2} + \frac{r[2k] b_\pm[k]}{\sigma^2} \\
& + \frac{r[2k+1] p_\pm[k]}{\sigma^2} + \overline{\alpha}^{(i)}[k-1] \Bigg) \\
- \max_{b_\pm[k]=0}{}^* \Bigg(& \overline{\beta}^{(j)}[k+1] \frac{L_e[k] \cdot b_\pm[k]}{2} + \frac{r[2k] b_\pm[k]}{\sigma^2} + \\
& + \frac{r[2k+1] p_\pm[k]}{\sigma^2} + \overline{\alpha}^{(i)}[k-1] \Bigg) \\
= L_e[k] + 2\frac{r[2k]}{\sigma^2} & + \max_{b_\pm[k]=1}{}^* \left(\overline{\beta}^{(j)}[k+1] + \frac{r[2k+1] p_\pm[k]}{\sigma^2} + \overline{\alpha}^{(i)}[k-1] \right) \\
& - \max_{b_\pm[k]=0}{}^* \left(\overline{\beta}^{(j)}[k+1] + \frac{r[2k+1] p_\pm[k]}{\sigma^2} + \overline{\alpha}^{(i)}[k-1] \right) \qquad \text{(A.32)}
\end{aligned}
$$

We give now the finishing touch if we assume that our vector $\mathbf{r}$ comes out of an AWGN channel with ideal matched-filter detection and no ISI, so that, with proper normalization, we have $\sigma^2 = N_0/(2E_s)$ (2.100). The conclusion is that

$$
\begin{aligned}
L[k] = L_e[k] & + \frac{4E_s}{N_0} r[2k] + \\
\Bigg\{ \max_{b[k]=1}{}^* & \left(\overline{\beta}^{(j)}[k+1] + \frac{2E_s}{N_0} r[2k+1] p_\pm[k] + \overline{\alpha}^{(i)}[k-1] \right) \\
- \max_{b[k]=0}{}^* & \left(\overline{\beta}^{(j)}[k+1] + \frac{2E_s}{N_0} r[2k+1] p_\pm[k] + \overline{\alpha}^{(i)}[k-1] \right) \Bigg\} \qquad \text{(A.33)}
\end{aligned}
$$

This final equation represents just what we intended to derive. It tells us that the LAPPR of the information bit $b[k]$ is made of three terms: the first one $(L_e[k])$ is something *extrinsic* to the detector, that is, *a priori* information provided by some external entitity. The second component, $r[2k] \cdot 4E_s/N_0$ is something directly related to the noisy output of the channel in correspondence of the information bit (recall that $a[2k] = b_\pm[k]$) and has nothing to do with coding/decoding. The third term (the one in braces) is what is actually computed by a modified BCJR algorithm (we will call it generically the Soft-Input, Soft-Output SISO detector) on the basis of the noisy parity bits $r[2k+1]$. This is the part that is peculiar to the detector and that can be made available at the exterior as "ouput" information that has been derived not using the prior extrinsic information $(L_e[k])$. With this interpretation, we can cast (A.33) into the following form:

$$
L[k] = L_{in}[k] + L_{ch}[k] + L_{out}[k] \qquad \text{(A.34)}
$$

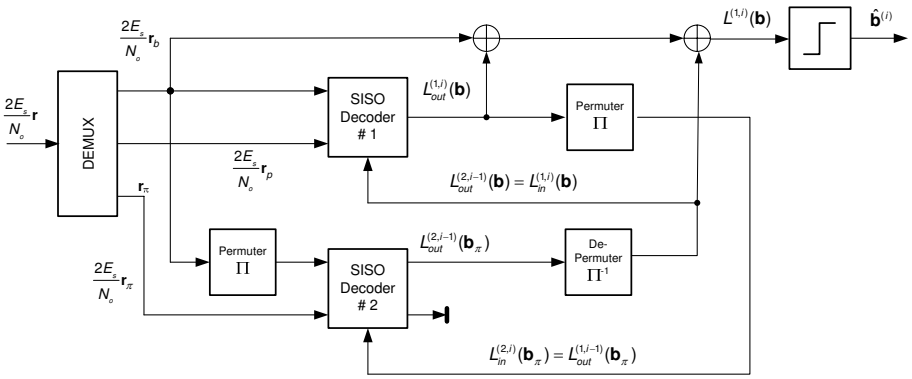


Figure A.2 Accurate representation of the functions of a parallel turbo decoder

where the different terms are easily understood, and where we see that the SISO detector actually computes $L_{out}[k]$ only.

The turbo effect of iterative decoding appears because of $L_{out}[k]$ actually being a function of $L_e[k]$ through $\overline{\gamma}^{(j,i)}[k]$ in (A.28) and (A.29), so that we can more accurately write

$$L[k] = L_{in}[k] + L_{ch}[k] + L_{out}\,[k; L_{in}[k]) \tag{A.35}$$

Based on (A.35), we can now eventually summarize with higher accuracy how turbo decoding proceeds: we have two further indices to be appended to (A.34): one $(1,2)$ identifies which of the two SISO decoders we are addressing, and the other (i) just indicates the generic decoding iteration number. At iteration $i, i = 1, \ldots, N_{it}$, decoder 1 computes

$$L_{out}^{(1,i)}\left[k; L_{in}^{(1,i)}[k]\right) \tag{A.36}$$

using the extrinsic information

$$L_{in}^{(1,i)}[k] \triangleq L_{out}^{(2,i-1)}[k] \tag{A.37}$$

so that the extrinsic information to decoder 1 is just the *output* LAPPR $L_{out}^{(2,i-1)}[k]$ computed by decoder 2 at the previous iteration. We know that it is extrinsic because it has been computed after parity bits that decoder 1 *does not* have available. In turn, decoder 2 computes

$$L_{out}^{(2,i)}\left[k; L_{in}^{(2,i)}[k]\right)$$

$$L_{in}^{(2,i)}[k] \triangleq L_{out}^{(1,i-1)}[k] \tag{A.38}$$

and so on. The decoding iterations start with $L_{out}^{(2,0)}[k] = 0$. The "channel" term $L_{ch}[k]$ can be actually added to L_{in} and L_{out} just only at the end of the decoding iterations, when the final hard-decision on $L^{(1,N_{it})}(b[k])$ is taken. A more accurate version of the turbo decoder of Fig. 5.15 (b) is shown in Fig. A.2, where the computation and the spinning of LAPPRs is explicitly shown.

APPENDIX B

THE ITERATIVE MESSAGE-PASSING ALGORITHM TO DECODE LDPC CODES

Once the conceptual aspect of iterative soft message passing has been presented in the main text, we develop here in detail such algorithm, starting from a nice simple property introduced by Gallager back in the 60s, whose relevance to a *parity check* code can be guessed. If we have a codeword $[a[0], a[1], ..., a[N-1]]$ whose independent symbols have (posterior) probabilities $[p_0, p_1, ..., p_{N-1}]$ of being equal to 1. (i.e., $p_n \triangleq \Pr\{a[n]=1|r[n]\}$, then the probability that we have an *even* number of 1s in that word is

$$\frac{1}{2}+\frac{1}{2}\prod_{n=0}^{N-1}(1-2p_n) \tag{B.1}$$

This can be easily shown by mathematical induction (exercise for the reader). When we observe the output of an AWGN channel $r[n] = a_{\pm}[n] + w[n]$, with equiprobable values for $a_{\pm}[n] \in \{-1, +1\}$, and variance $\sigma^2 = (2E_s/N_0)^{-1}$ for $w[n]$, then the posterior probabilities above are

$$p_n \triangleq \Pr\{a_{\pm}[n]=1|r[n]\} = \frac{f_r(r[n]|a_{\pm}[n]=1)\Pr\{a_{\pm}[n]=1\}}{f_r(r[n])}$$

$$= \frac{\exp[-(r[n]-1)^2/2\sigma^2]\cdot 0.5}{0.5\exp[-(r[n]-1)^2/2\sigma^2]+0.5\exp[-(r[n]+1)^2/2\sigma^2]}$$

$$= \frac{\exp(r[n]/\sigma^2)}{\exp(r[n]/\sigma^2) + \exp(-r[n]/\sigma^2)} = \frac{1}{1 + \exp(-2r[n]/\sigma^2)}$$

$$= \frac{1}{1 + \exp(-4r[n]E_s/N_0)} \tag{B.2}$$

where we considered that the unconditional pdf $f_r(x)$ of $r[n]$ is an equiprobable mixture of two Gaussian pdfs with modes (conditional means) ± 1 and variance σ^2.

Now, we intend to compute an a-posteriori probability ratio (APPR) of the kind

$$\frac{\Pr\left\{a[n] = 1 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}}{\Pr\left\{a[n] = 0 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}} \tag{B.3}$$

that appears in (B.1). Using Bayes' rule twice, the APPR above can be reformulated as

$$\frac{p_n \cdot \Pr\left\{ \{c_i = 0\}_{i \in C_{i,n}} \middle| a[n] = 1, \mathbf{r}\right\}}{(1 - p_n) \cdot \Pr\left\{ \{c_i = 0\}_{i \in C_{i,n}} \middle| a[n] = 0, \mathbf{r}\right\}} \tag{B.4}$$

Now, if $a[n] = 0$, satisfying the i-th parity check equation means that the other bits in each check equation must bear an *even* number of 1s. We can compute this probability by suited modification of (B.1):

$$\frac{1}{2} + \frac{1}{2} \prod_{n', i \in R_{i \backslash n}} (1 - 2p_{n',i}) \tag{B.5}$$

where $R_{i \backslash n}$ represents the set of locations with entries equal to 1 in the i-th column of the parity-check matrix, excluding location n, and where the index n', i addresses the n'-th bit in the i-th parity check equation. A similar reasoning applies of course to $a[n] = 1$. Also, assuming that the $a[n]$ are mutually independent, the probability that *all* parity checks are verified is just the product of those probabilities. Therefore, combining (B.4) and (B.5) and considering the latter remark, we get

$$\frac{\Pr\left\{a[n] = 1 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}}{\Pr\left\{a[n] = 0 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}} = \frac{p_n \cdot \prod_{i \in C_{i,n}} \left(1 - \prod_{n', i \in R_{i \backslash n}} (1 - 2p_{n',i})\right)}{(1 - p_n) \cdot \prod_{i \in C_{i,n}} \left(1 + \prod_{n', i \in R_{i \backslash n}} (1 - 2p_{n',i})\right)} \tag{B.6}$$

This is nice, but cumbersome to compute as it stands - it is time now to introduce the *iterative algorithm* based on the exchange of messages on the Tanner graph. Let us try to relate the messages $\rho_{i,n}$ and $q_{n,i}$ introduced in the main text with the probabilities we have just computed. First, we recognize from (5.54)

$$\rho_{i,n}(0) = \frac{1}{2} + \frac{1}{2} \prod_{n', i \in R_{i \backslash n}} (1 - 2p_{n',i})$$

$$\rho_{i,n}(1) = 1 - \rho_{i,n}(0) = \frac{1}{2} - \frac{1}{2} \prod_{n', i \in R_{i \backslash n}} (1 - 2p_{n',i}) \tag{B.7}$$

so that

$$\frac{\Pr\left\{a[n] = 1 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}}{\Pr\left\{a[n] = 0 | \mathbf{r}, \{c_i = 0\}_{i \in C_{i,n}}\right\}} = \frac{p_n \prod_{i \in C_{i,n}} \rho_{i,n}(1)}{(1 - p_n) \prod_{i \in C_{i,n}} \rho_{i,n}(0)} \tag{B.8}$$

In addition to this, recalling (5.53), we have

$$q_{n,i}(1) = (1-p_n) \prod_{i' \in C_{i' \setminus i,n}} \rho_{i',n}(1)$$

$$q_{n,i}(0) = p_n \prod_{i' \in C_{i' \setminus i,n}} \rho_{i',n}(0) \tag{B.9}$$

so that now we are almost ready to compute all of the messages that are exchanged on the Tanner graph of the LDPC as described in the main text: we only miss the relation to compute ρ from q: we identify $q_{n',i}$ with $p_{n',i}$ in (B.5) so that we can relate the ρ's to the q's through (B.7).

To recap and re-frame in good order, the iterative message passing algorithm does this:

1. *Initialization*:
   Set the iteration number $m = 0$ and $\forall i$ do

$$q_{n,i}^{(0)}(1) = \frac{1}{1+\exp(-4r[n]E_s/N_0)}$$

$$q_{n,i}^{(0)}(0) = \frac{1}{1+\exp(4r[n]E_s/N_0)} \tag{B.10}$$

   This is the values of initial posteriori probabilities (B.2) when we don't (yet) have any information available from checknodes.

2. *Exchange of messages*:
   Update ρ:

$$\rho_{i,n}^{(m+1)}(0) = \frac{1}{2} + \frac{1}{2} \prod_{n' \in R_{i \setminus n}} (1 - 2q_{n',i}^{(m)}(0))$$

$$\rho_{i,n}^{(m+1)}(1) = 1 - \rho_{i,n}^{(m+1)}(0) \tag{B.11}$$

   Update q:

$$q_{n,i}^{(m+1)}(0) = A_{n,i}^{(m+1)} \cdot (1-p_n) \prod_{i' \in C_{i' \setminus i,n}} \rho_{i',n}^{(m)}(0)$$

$$q_{n,i}^{(m+1)}(1) = A_{n,i}^{(m+1)} \cdot p_n \prod_{i' \in C_{i' \setminus i,n}} \rho_{i',n}^{(m)}(1) \tag{B.12}$$

   where $A_{n,i}^{(m+1)}$ is a normalization constant so as to always ensure that $q_{n,i}^{(m+1)}(0) + q_{n,i}^{(m+1)}(1) = 1$ (they are probabilities!)[1]

3. *Hard-Decision*:
   Compute the a-posteriori probabilities of the coded bits $a[n]$:

$$Q_n^{(m+1)}(0) = A_n^{(m+1)} \cdot (1-p_n) \prod_{i \in C_{i,n}} \rho_{i,n}^{(m)}(0)$$

[1] In the computation of $q_{n,i}^{(m+1)}(x)$ we could have also used the just refreshed values of $\rho_{i,n}^{(m+1)}(x)$ - it is a design choice often related to a specific HW architecture

$$Q_n^{(m+1)}(1) = A_n^{(m+1)} \cdot p_n \prod_{i \in C_{i,n}} \rho_{i,n}^{(m)}(1) \tag{B.13}$$

(where again $A_n^{(m+1)}$ is a normalization constant) and then perform the tentative decision

$$\hat{a}^{(m+1)}[n] = \begin{cases} 1 & Q_n^{(m+1)}(1) > 0.5 \\ 0 & \text{else} \end{cases} \tag{B.14}$$

4. *Termination*:
If $\hat{\mathbf{a}}^{(m+1)}\mathbf{H}^T = \mathbf{0}$, or $(m+1)$ is equal to the maximum number of allowed iterations N_{it}, then stop iterating and output the current value $\hat{\mathbf{a}}^{(m+1)}$ as your estimated codeword. Else, increment m and goto 2. to perform a new iteration.

As is seen, we have added the superscript $^{(m)}$ to explicitly address the iteration number in the development of the algorithm.

The decoders that are actually used in the practice, as happens with turbo codes, are actually based on a procedure than operates in the log-probability domain and turns out to be much more numerically stable. As stated before, the log-domain decoder operates of LAPPRs instead of probabilities. We have already introduced $L(\rho_{i,n})$ and $L(q_{n,i})$ in the main text, so we will not insist on them again here. Before introducing the new "item list" for the log-message passing, we need a few more computations.

First, we observe that

$$\ln\left(\frac{1+e^x}{1+e^{-x}}\right) = \ln\left(\frac{e^{x/2}(e^{-x/2}+e^{x/2})}{e^{-x/2}(e^{x/2}+e^{-x/2})}\right) = x \tag{B.15}$$

so that, at $m = 0$,

$$L^{(0)}(q_{n,i}) = \ln\left(\frac{1+\exp(4r[n]E_s/N_0)}{1+\exp(-4r[n]E_s/N_0)}\right) = \frac{4E_s}{N_0} r[n] \tag{B.16}$$

as we already knew from the main text. Then, the most difficult step is working out an expression of $L(\rho_{i,n})$ as a function of $L(q_{n,i})$. We start from (B.7) that we repeat hereafter:

$$\rho_{i,n}(0) = \frac{1}{2} + \frac{1}{2} \prod_{n' \in R_{i \setminus n}} (1 - 2p_{n',i})$$

$$\rho_{i,n}(1) = 1 - \rho_{i,n}(0) = \frac{1}{2} - \frac{1}{2} \prod_{n' \in R_{i \setminus n}} (1 - 2p_{n',i}) \tag{B.17}$$

and we observe that if we have two probabilities p_0 and p_1 such that $p_0 + p_1 = 1$, then we have

$$p_0 - p_1 = 1 - 2p_1 = \frac{p_0 - p_1}{p_0 + p_1} = \frac{1 - p_1/p_0}{1 + p_1/p_0} = \frac{\sqrt{p_0/p_1} - \sqrt{p_1/p_0}}{\sqrt{p_0/p_1} + \sqrt{p_1/p_0}}$$

$$= \frac{\exp(\ln\sqrt{p_0/p_1}) - \exp(-\ln\sqrt{p_0/p_1})}{\exp(\ln\sqrt{p_0/p_1}) + \exp(-\ln\sqrt{p_0/p_1})} = \tanh\left(\frac{1}{2}\ln(p_0/p_1)\right) \tag{B.18}$$

and the log-probability ratio is

$$\ln(p_1/p_0) = -2\tanh^{-1}(1 - 2p_1)$$

Using the relation above and (B.17), we get

$$L(\rho_{i,n}) = -2\tanh^{-1}\prod_{n'\in R_{i\setminus n}}(1-2p_{n',i}) = -2\tanh^{-1}\prod_{n'\in R_{i\setminus n}}\tanh\left(-\frac{L(q_{n',i})}{2}\right) \tag{B.19}$$

We can go on elaborating this to try to obtain an expression that needs no products to be computed. Recalling that $\tanh$ is an *odd* and *strictly monotonic* increasing function, we can write

$$\begin{aligned} L(\rho_{i,n}) &= 2\tanh^{-1}\left(\prod_{n'\in R_{i\setminus n}}\tanh(|L(q_{n',i})|/2)\left\{-\prod_{n'\in R_{i\setminus n}}\operatorname{sgn}[-\tanh(L(q_{n',i}))]\right\}\right) \\ &= 2\tanh^{-1}\left(\prod_{n'\in R_{i\setminus n}}\tanh(|L(q_{n',i})|/2)\right)\cdot\left\{-\prod_{n'\in R_{i\setminus n}}\operatorname{sgn}[-L(q_{n',i})]\right\} \end{aligned} \tag{B.20}$$

so that we have separated the computation of the *amplitude* and of the *sign* of $L(\rho_{i,n})$. The amplitude reads

$$\begin{aligned} |L(\rho_{i,n})| &= 2\tanh^{-1}\exp\left[\ln\left(\prod_{n'\in R_{i\setminus n}}\tanh(|L(q_{n',i})|/2)\right)\right] \\ &= 2\tanh^{-1}\exp\left[\left(\sum_{n'\in R_{i\setminus n}}\ln\tanh(|L(q_{n',i})|/2)\right)\right] \end{aligned} \tag{B.21}$$

that can be further simplified introducing the composite function

$$\phi(\alpha) \stackrel{\triangle}{=} -\ln\tanh(\alpha/2) \tag{B.22}$$

whose shape is shown in Fig. B, so that

$$|L(\rho_{i,n})| = \phi^{-1}\left[\sum_{n'\in R_{i\setminus n}}\phi\left(|L(q_{n',i})|\right)\right] \tag{B.23}$$

The nice property of function ϕ is that (for $\alpha > 0$) $\phi(\phi(\alpha)) = \alpha$ (or, in other words, $\phi = \phi^{-1}$). Using such property, the log-probability ratio for $\rho_{i,n}$ finally becomes

$$L(\rho_{i,n}) = \left\{-\prod_{n'\in R_{i\setminus n}}\operatorname{sgn}[-L(q_{n',i})]\right\}\cdot\phi\left[\sum_{n'\in R_{i\setminus n}}\phi(|L(q_{n',i})|)\right] \tag{B.24}$$

After this has been painfully done, the expressions for $L(q_{n,i})$ and $L(Q_n)$ are trivial to compute (notice that the normalization constants all disappear in the ratio!), so that we are ready to conclude with the new list of steps for *log-domain decoding*:

1. *Initialization*:
   Set $m = 0$ and
$$L^{(0)}(q_{n,i}) = \frac{4E_s}{N_0}r[n] \tag{B.25}$$

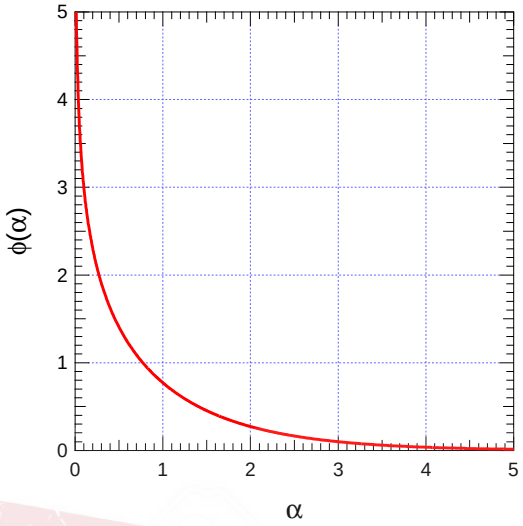


Figure B.1 Function $\phi(\alpha)$ defined in the text

2. *Exchange of messages*:
   Update $L(\rho)$:

$$L^{(m+1)}(\rho_{i,n}) = -\prod_{n' \in R_{i \setminus n}} \text{sgn}[-L^{(m)}(q_{n',i})] \cdot \phi\left[\sum_{n' \in R_{i \setminus n}} \phi(|L^{(m)}(q_{n',i})|)\right] \quad \text{(B.26)}$$

   Update of $L(q)$:

$$L^{(m+1)}(q_{n,i}) = \frac{4E_s}{N_0} r[n] + \sum_{\substack{i' \neq i \\ i' \in C_{i',n}}} L^{(m)}(\rho_{i',n}) \quad \text{(B.27)}$$

3. *Hard-Decision*:
   Compute the LAPPR of the coded symbols $a[n]$:

$$L^{(m+1)}(Q_n) = \frac{4E_s}{N_0} r[n] + \sum_{i \in C_{i,n}} L^{(m)}(\rho_{i,n}) \quad \text{(B.28)}$$

   then perform the tentative decision

$$\hat{a}^{(m+1)}[n] = \text{sgn}[L^{(m+1)}(Q_n)] \quad \text{(B.29)}$$

4. *Termination*:
   If $\hat{\mathbf{a}}^{(m+1)}\mathbf{H}^T = 0$ or $(m+1) = N_{it}$, then stop iterating and output $\hat{\mathbf{a}} = \hat{\mathbf{a}}^{(m+1)}$ as your estimated codeword. Else, increment m and goto 2. to perform a new iteration.

We can give a further finishing touch to message passing in the log-domain to come to what is usually used in hardware to decode LDPC codes, namely, the so called *normalized-min-sum* algorithm (or simply min-sum). Looking at Fig. B, we can derive a simple

approximation of the update rule (B.33) for $L(\rho)$. Here, we are requested to compute the sum $\sum_{n' \in R_{i \setminus n}} \phi(|L^{(m)}(q_{n',i})|)$ of a few values of the function $\phi(\alpha)$. Such function is highly nonlinear in its argument α and in particular it is very fast decreasing in α. We may conjecture that the only term in the sum that really matters is the one that we get in respect of the *minimum* among the values of the arguments $|L^{(m)}(q_{n',i})|$. This means saying that

$$\phi\left[\sum_{n' \in R_{i \setminus n}} \phi(|L^{(m)}(q_{n',i})|)\right] \simeq \phi\left[\phi(\min_{n' \in R_{i \setminus n}} |L^{(m)}(q_{n',i})|)\right]$$

but $\phi(\phi(\alpha)) = \alpha$, so we have

$$\phi\left[\sum_{n' \in R_{i \setminus n}} \phi(|L^{(m)}(q_{n',i})|)\right] \simeq \min_{n' \in R_{i \setminus n}} |L^{(m)}(q_{n',i})| \tag{B.30}$$

and we do not need any device to actually compute the function ϕ. The rule for the update of $L(\rho)$ eventually reads

$$L^{(m+1)}(\rho_{i,n}) = - \prod_{n' \in R_{i \setminus n}} \operatorname{sgn}[-L^{(m)}(q_{n',i})] \cdot \min_{n' \in R_{i \setminus n}} |L^{(m)}(q_{n',i})| \tag{B.31}$$

This approximation entails a small performance degradation in terms of BER that turns out to be negligible with respect to the considerable simplification in the implementation of the decoder. The interpretation of this update equation is given in the main text under equaiton (5.57.

Finally, we can also *normalize* the min-sum algorithm by simply dropping the factor $4E_s/N_0$ in the initialization of $L(q)$ (B.32) and in all subsequent iterative updates (B.34) and (B.35) with no further performance loss, obtaining the so-called *normalized min-sum* algorithm. Normalization is particularly expedient since it allows to implement a decoder that, like the Viterbi Algorithm for MLSE of convolutional codes, does not need any side information about the signal-to-noise ratio. The pseudo-code of the (normalized)-min-sum algorithm is finally the following:

1. *Initialization*:
   Set $m = 0$ and
   $$L^{(0)}(q_{n,i}) = r[n] \tag{B.32}$$

2. *Exchange of messages*:
   Update $L(\rho)$:
   $$L^{(m+1)}(\rho_{i,n}) = - \prod_{n' \in R_{i \setminus n}} \operatorname{sgn}[-L^{(m)}(q_{n',i})] \cdot \min_{n' \in R_{i \setminus n}} |L^{(m)}(q_{n',i})| \tag{B.33}$$

   Update of $L(q)$:
   $$L^{(m+1)}(q_{n,i}) = r[n] + \sum_{\substack{i' \neq i \\ i' \in C_{i',n}}} L^{(m)}(\rho_{i',n}) \tag{B.34}$$

3. *Hard-Decision*:
   Compute the LAPPR of the coded symbols $a[n]$:

$$L^{(m+1)}(Q_n) = r[n] + \sum_{i \in C_{i,n}} L^{(m)}(\rho_{i,n}) \tag{B.35}$$

   then perform the tentative decision

$$\hat{a}^{(m+1)}[n] = \text{sgn}[L^{(m+1)}(Q_n)] \tag{B.36}$$

4. *Termination*:
   If $\hat{\mathbf{a}}^{(m+1)}\mathbf{H}^T = 0$ or $(m+1) = N_{it}$, then stop iterating and output $\hat{\mathbf{a}} = \hat{\mathbf{a}}^{(m+1)}$ as your estimated codeword. Else, increment m and goto 2. to perform a new iteration.

APPENDIX C

SUCCESSIVE-CANCELATION DECODING OF POLAR CODES

Assume as usual that the encoded bits $a[n]$ are remapped and sent out over an AWGN channel, so that the *soft* observed vector is $\mathbf{r} = \mathbf{a}_{\pm} + \mathbf{w}$ as in LDPC decoding, playing the role of the *hard* output $\mathbf{d}$ of the binary vector channel. The SCD algorithm is by nature bit-recursive, and stems of course out of the recursive construction of the encoder.

Let us start with the decoding algorithm for the order-2 encoder in Fig. 5.33. We regard the decoder as a Tanner graph to be crossed in the reverse order wrt to the encoder, starting from the values of $r[0]$ and $r[1]$, and we try to find the values of the messages produced by the nodes in the log-domain of LLRs, as was done for the LDPC decoder in Appendix 2.3.

We start by observing that

$$f(r[0], r[1] \mid c[0], c[1]) = f(r[0] \mid c[0] \oplus c[1]) \cdot f(r[1] \mid c[1])$$

and

$$f(r[0], r[1], c[1] \mid c[0]) = f(r[0] \mid c[0] \oplus c[1]) \cdot f(r[1] \mid c[1]) \cdot p\,(c[1])$$

where as usual $f(\cdot)$ indicates the pdf, and $p(\cdot)$ the probability mass function, so that, if we marginalize,

$$f(r[0], r[1] \mid c[0]) = \frac{1}{2} \sum_{c[1]} f(r[0] \mid c[0] \oplus c[1]) \cdot f(r[1] \mid c[1]) \tag{C.1}$$

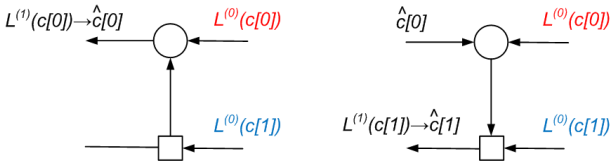


Figure C.1 Two-step 2×2 decoding

Assuming AWGN channel, from (C.1) we can find the value of the LLR for the input bit $c[0]$ as follows:

$$L^{(1)}(c[0]) \stackrel{\triangle}{=} \log \frac{f(r[0], r[1] \mid c[0] = 1)}{f(r[0], r[1] \mid c[0] = 0)}$$

$$= -2 \tanh^{-1} \left(\tanh \left(\frac{(L^{(0)}(c[0])}{2} \right) \tanh \left(\frac{(L^{(0)}(c[1])}{2} \right) \right)$$

$$\simeq -\text{sgn} \left(L^{(0)}(c[0]) \right) \text{sgn} \left(L^{(0)}(c[1]) \right) \min \left(|L^{(0)}(c[0])|, |L^{(0)}(c[1])| \right) \tag{C.2}$$

where the two LLRs values $L^{(0)}(c[0])$ and $L^{(0)}(c[1])$ derived from the soft outputs $r[0]$ and $r[1]$, respectively, of the AWGN channel are:

$$L^{(0)}(c[n]) = 4E_s/N_0 \cdot r[n] \quad n = 0, 1 \tag{C.3}$$

The superscripts in the values of the LLRs indicate the stage (m) of decoding: in this example, $m = 0$ means input and $m = 1$ means first (and last) stage of computation.

Once $L^{(1)}(c[0])$ is found as above, we can proceed to perform decision on $c[0]$ (based on the observation of $r[0]$ and $r[1]$ only) considering the sign of the LLR. Now comes the second step of decoding, involving cancelation of the previously (just) decoded bit $\hat{c}[0]$: we can compute

$$L^{(1)}(c[1]) = \stackrel{\triangle}{=} \log \frac{f(r[0], r[1] \mid \hat{c}[0], c[1] = 1)}{f(r[0], r[1] \mid \hat{c}[0], c[1] = 0)}$$

$$= \log \frac{f(r[0] \mid \hat{c}[0] \oplus c[1] = 1) \cdot f(r[1] \mid c[1] = 1)}{f(r[0] \mid \hat{c}[0] \oplus c[1] = 0) \cdot f(r[1] \mid c[1] = 0)}$$

$$= \text{sgn}\,(\hat{c}[0]) \cdot L^{(0)}(c[0]) + L^{(0)}(c[1]) = \frac{4E_s}{N_0} \left(\text{sgn}\,(\hat{c}[0]) \cdot r[0] + r[1] \right) \tag{C.4}$$

Notice that in general $\text{sgn}\,(\hat{c}[0]) \cdot L^0(c[0]) \neq |L^0(c[0])|$ since $\hat{c}[0]$ is based on the observation of $r[0]$ *and* $r[1]$, whilst $L^0(c[0])$ is based on $r[0]$ only. The procedure is summarized and sketched in Fig. C that represents the order-2 decoder devised from the encoder, assuming that now processing goes from right to left and that the inputs are the values of the two channel LLRs $L^0(c[0])$ and $L^{(0)}(c[1])$. The nodes represents processors that either forward inputs or compute equations (C.2)-(C.4). In the decoding steps above, if one of the $c[n]$ to be estimated turns out to be frozen and then known, there is clearly no need to actually carry out the relevant computation.

How can we extend to order-4 decoding? We refer to the recursive construction of the encoder, trying to "invert" it to get to a decoding algorithm. Looking at Fig. C, we see that starting form the rightmost inputs $L^{(0)}(c[n])$, $n = 0, 1, 2, 3$ we can compute the two LLRs $L^{(1)}(c[0])$ and $L^{(1)}(c[1])$ by combining the offset-2 inputs from the channel as follows:

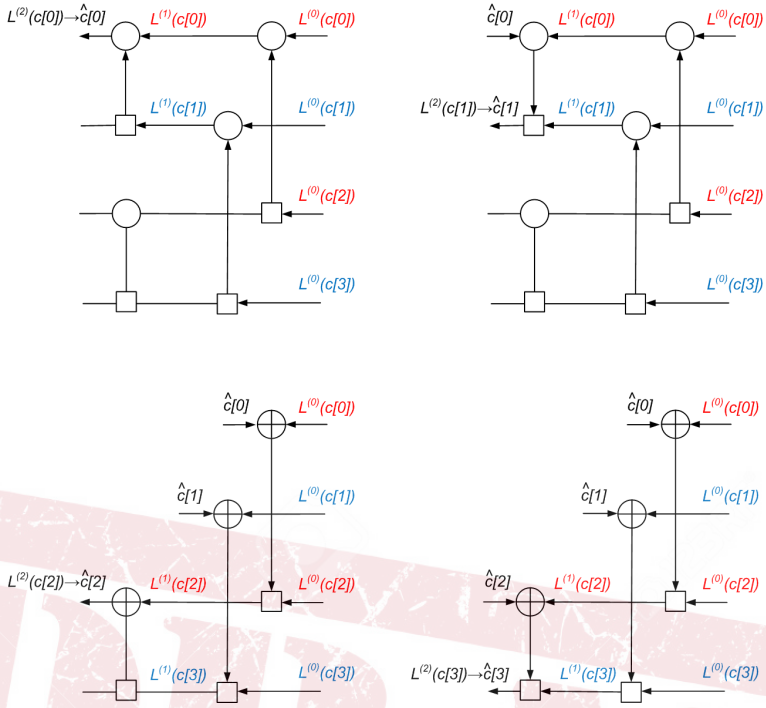


Figure C.2 Order-4 polar decoding

$$L^{(1)}(c[0]) = -2\tanh^{-1}\left(\tanh\left(\frac{(L^{(0)}(c[0])}{2}\right)\tanh\left(\frac{(L^{(0)}(c[2])}{2}\right)\right)$$

$$L^{(1)}(c[1]) = -2\tanh^{-1}\left(\tanh\left(\frac{(L^{(0)}(c[1])}{2}\right)\tanh\left(\frac{(L^{(0)}(c[3])}{2}\right)\right) \tag{C.5}$$

After this is done, we can perform usual order-2 decoding as above, starting form the values of $L^{(1)}(c[0])$ and $L^{(1)}(c[1])$ as inputs, to get the two values of $\hat{c}[0]$ (after the observation of *all* $L^{(0)}(c[n])$) and $\hat{c}[1]$ (after the observation of all $L^{(0)}(c[n])$ *and* of $\hat{c}[0]$). Then we move to the lower part of the decoder: we compute the relevant interference-canceled inputs

$$L^{(1)}(c[2]) = L^{(0)}(c[2]) + \text{sgn}\,(\hat{c}[0]) \cdot L^{(0)}(c[0])$$

$$L^{(1)}(c[3]) = L^{(0)}(c[3]) + \text{sgn}\,(\hat{c}[1]) \cdot L^{(0)}(c[1]) \tag{C.6}$$

and conclude with another application of order-2 decoding to eventually derive $\hat{c}[2]$ and $\hat{c}[3]$. The same *divide-and-conquer* reasoning also works for order-8: first find $L^{(1)}(c[n])$, $n = 0, 1, 2, 3$ by preprocessing the pairs $\left(L^{(0)}(c[n]), L^{(0)}(c[n+4])\right)$, $n = 0, 1, 2, 3$ to find $\hat{c}[0], \hat{c}[1], \hat{c}[2], \hat{c}[3]$ with order-4 decoding, then compute the interference-canceled LLRs $L^{(1)}(c[n])$, $n = 4, 5, 6, 7$, and eventually estimate from the latter $c[4], c[5], c[6], c[7]$ via another order-4 decoding. By iterating this reasoning, we can get as far as we wish in the blocklength according to the following pseudocode: `Decode`$(N; L_0, \ldots, L_{N-1})$:
If $N = 2$, then decode according to (C.3-C.2-C.4) and stop; else:

1. *Combination*:
   for $i = 0, \ldots, N/2 - 1$ compute

$$L^{(1)}(c[i]) = -2 \tanh^{-1} \left(\tanh \left(\frac{(L^{(0)}(c[i])}{2} \right) \tanh \left(\frac{(L^{(0)}(c[i + N/2])}{2} \right) \right)$$

2. *First sub-decoding*:
   $\texttt{Decode}(N/2; L^{(1)}(c[0]), \ldots, L^{(1)}(c[N/2 - 1])) \rightarrow \hat{c}[0], \ldots, \hat{c}[N/2 - 1]$

3. *Interference Cancellation*:
   per $i = 0, \ldots, N/2 - 1$ calcolare

$$L^{(1)}(c[N/2 + i]) = L^{(0)}(c[N/2 + i]) + \text{sgn}\,(\hat{c}[i]) \cdot L^{(0)}(c[i])$$

4. *Second sub-decoding*:
   $\texttt{Decode}(N/2; L^{(1)}(c[N/2]), \ldots, L^{(1)}(c[N - 1])) \rightarrow \hat{c}[N/2], \ldots, \hat{c}[N - 1]$

CHAPTER 4

ANYWHERE, ANYTIME - WIRELESS COMMUNICATIONS FUNDAMENTALS

”There is no distance on Earth that radio communications cannot conquer.”
—Guglielmo Marconi, 1901, Comment to his own successful first trans-Atlantic radio communication experiment

Figure 4.1 Satellite connection with parabolic (directive) antenna

One of the dreams of the communications engineer is that of designing a good signaling scheme for transmission of high bit-rate signals over the wireless channel. That is, being robust enough to withstand the formidable distortion experienced by the radio signal during propagation into the multipath channel - just like building a signal that looks *wideband* and *narrowband* at the same time... The solution is *multicarrier modulation*: placing many *narrowband* data signals side by side within the (wide) transmission bandwidth, on adjacent (sub)carriers, and in such a way that each one no longer experiences distortion from the channel: a rainbow of data of different *colors* that make the dream come true.

4.1 The Wireless Communication Channel

Up to now, we have mainly taken into consideration examples of signal detection on the AWGN channel. This is typical of communications with a geostationary[1] satellite with a directional antenna that is aimed straight at the satellite (think of direct-to-home digital satellite television as in Fig. 4.1), wherein the model of the digital signal at the receiving antenna is relatively simple:

$$x(t) = a\exp(\jmath\phi)\cdot s(t-\tau) + w(t) \tag{4.1}$$

where we have assumed that $s(t)$ is unit-power: $\mathrm{E}\{|s(t)|^2\} = 1$.

In (4.1), $a = \sqrt{2P_R}$ is the *amplitude* of the received signal so that the received (radio) power is P_R, ϕ is the *phase* shift experienced by the carrier during propagation, τ is the group *delay*, i.e., the delay experienced during the propagation by the (complex) envelope of the data signal, and $w(t)$ is the usual AWGN term with I/Q independent components both having a power spectral density N_0. The parameters a, ϕ, τ are in general unknown to the receiver, and has to be estimated before performing digital detection with the data demodulator in Fig. 2.29, so that the signal constellation is perfectly restored.

In terrestrial radio communications things get a bit more complicated since the antennas that are used in terrestrial terminals, often used for mobile communications, are *omnidirectional* i.e., they receive radio signals from any (horizontal) direction on the horizon just to be able to keep the link alive in spite of motion - this situation needs a more accurate

[1] meaning that the satellite appears to the Earth receiver as still in the sky

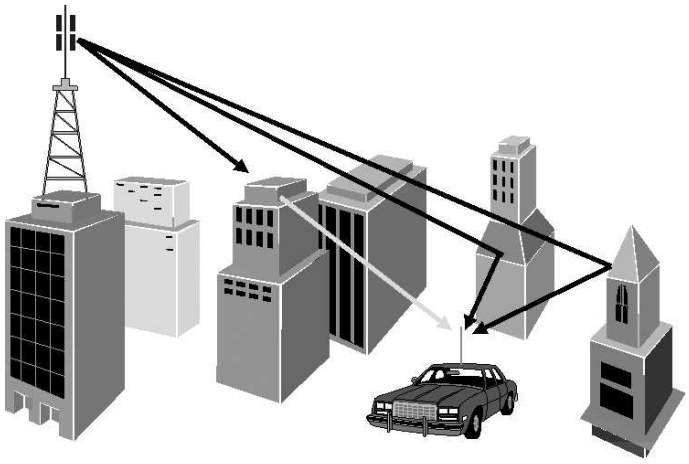


Figure 4.2 Sketch of the multipath radio channel

modeling than before (for more details on directionality of antennas, read sect. 6.3.1 and come back).

In the following, we will present a simplified approach to the modeling of a (terrestrial) wireless radio channel that is sufficient for the analysis of most practical cases. The general time-frequency analysis of random time-varying radio channels is considered too advanced to be presented and understood in the limited amount of pages that we will use here.

4.1.1 The multipath radio channel

The assumption made for the AWGN channel (4.1) is that, apart from background noise, what we get at the receiving end is a scaled and delayed replica of $s(t)$ directly coming form the satellite TX antenna, the scaling factor depending from the so-called "free-space attenuation" of radio waves, plus an unknown carrier phase shift.

Using and omnidirectional antenna means, as outlined above, that the transmit and receive antennas scatter and collect the radio signal into and from all directions on the horizontal plane, respectively. It may also happen that the antennas are surrounded by a number of hills, buildings, trees, that act as *reflectors* of the radio waves. Such situation is depicted in Fig. 4.2 where we outlined the *propagation paths* of the radio wave from the transmitter to the receiver.

From electromagnetic waves theory it is known that the propagation of a radio wave can be described as if occurring along a series of *rays*, just like with light, provided that the physical size of the objects making up the environment wherein propagation takes place is much larger then the wavelength λ_0 of the radio wave. Such description is that of *geometrical optics*. Focusing on radio signals with a carrier frequency f_0 close to 1 GHz (or higher), we have λ_0=30 cm (or smaller), so that our hypothesis is well verified and our reasoning is valid. The first conclusion of our analysis is that the received signal is no longer made of a single replica of $s(t)$. On the contrary, $s(t)$ travels along a multitude of different propagation $paths$ corresponding to the different rays of geometrical optics. Such rays are collected by the receiving antenna that gives the following signal as the result of *multipath*

propagation:

$$x(t) = \sum_{i=1}^{N_R} a_i \exp(\jmath\phi_i) \cdot s(t-\tau_i) = \sum_{i=1}^{N_R} h_i \cdot s(t-\tau_i) \tag{4.2}$$

where $h_i = a_i \mathrm{e}^{\jmath\phi_i}$ - a straightforward generalization of (4.1).

Although simplified, this model captures the fundamental effect of multipath. We see that the signal $x(t)$ collected by the receiving antenna is made of N_R "copies" of the transmitted signal, where N_R is the number of propagation paths that can be identified (or, technically speaking, *resolved*). Each path is characterized by three parameters: the *amplitude* $a_i > 0$ with which the signal is received on path #i, the *phase shift* ϕ_i the radio wave carrier undergoes while propagating on the same path, and finally the path delay τ_i experienced by the baseband components of the modulated wave (the so-called *group delay*). Assuming that the locations of the antennas are known, such parameters can be measured, or they can be computed to a good accuracy by a *ray-tracing* simulation program applied to a digital description of the geographic environment.

What is the effect of multipath on the detection of data signals? Let us disregard for the moment the channel noise $w(t)$ that anyway accompanies the received signal $x(t)$. Assume also that we can set in (4.2)

$$a_1 = 1 \ , \ \ \phi_1 = 0, \ \ \tau_1 = 0 \tag{4.3}$$

This means that we are taking the propagating path #1 as our reference, and that the receiver is somehow capable of *synchronizing* (locking) onto it, and estimating/compensating for the relevant parameters. In this case, (4.2) changes into

$$x(t) = s(t) + \sum_{i=2}^{N_R} h_i \cdot s(t-\tau_i) = s(t) + x_{MP}(t) \tag{4.4}$$

where usually $\tau_2 < \tau_3 < ... < \tau_{N_R}$. We may be tempted to use a conventional matched-filter detector to recover the data in $s(t)$. If we do so, we see that output of the matched filter is accompanied by an additional term, the result of filtering of $x_{MP}(t)$, that is produced by multipath. Such term causes interference on the useful component, much similar to what happened in (4.2) with ISI on a bandlimited channel. We have to expect a noticeable degradation of the BER performance of the receiver with respect to the matched filter bound.

It is also simple to recognize that the fundamental relation (4.2) that describes the multipath channel can be cast into the formulation of an LTI system whose impulse response is just

$$h_C(t) = \sum_{i=1}^{N_R} a_i \exp(\jmath\phi_i) \cdot \delta(t-\tau_i) \tag{4.5}$$

where each impulse function "marks" the presence of an echo. A partial representation of $h_C(t)$ is as in Fig. 4.3 (a), where we can easily appreciate the delays, the amplitudes (given as different amplitude of the δ's, whatever it may mean), and the phases of the rays. Note that the physical CIR if normalized as in (4.3). A more popular way to graphically characterize the physical CIR is shown in Fig. Fig. 4.3 (b), where we plot what is called the *delay spectrum* or the *power profile* of the channel. The x-axis represents a range of the delay of the possible echoes (rays) generated by the channel, and a bar is placed where an echo is actually present. The height of the bar is proportional to the power associated to that

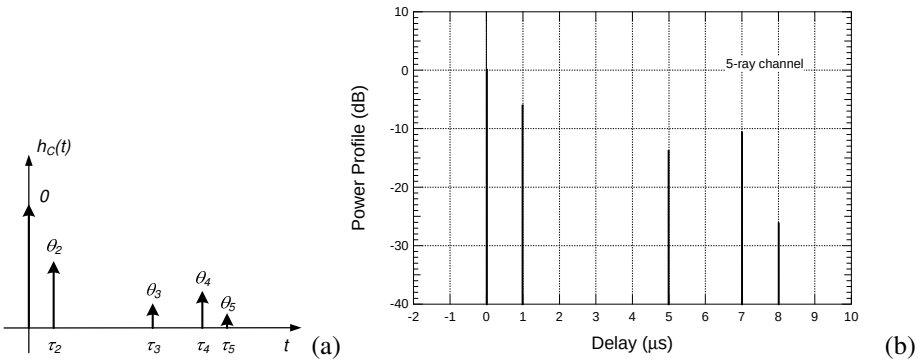


Figure 4.3 Impulse response (a) and Power Profile of a 5-path Multipath Channel (b)

path, and is often reported in dB wrt to the amplitude of the "main" path. No information is given on the phases of the propagation paths, for a good reason that we will discuss later on.

In some cases. it may happen that we do not actually succeed in identifying well-located echoes easily resolvable one from the other. In a sense, we could say that, instead of having a few "concentrated" echoes that appear in the physical CIR as in like in Fig. 4.3 (a), we have a multitude of smaller echoes that cram onto each other, so that the spiky CIR smears into a *continuum* of components that is better described by a *continuous* CIR $h_C(t)$ like with any ordinary LTI system, and the multipath characterization (4.2) just collapses into a standard convolution integral of $s(t)$ with $h_C(t)$ - we will say more about this in Sect. 4.1.4. Were we dealing with audio signal propagating in an environment, our multipath model (4.4) would correspond to what we experience when strolling onto an high-mountain trail and shouting to the leader of the row: isolated, well-identifiable echoes produced by the mountain peaks nearby. The continuous CIR would on the contrary characterize the reverberation effect experienced inside a large cathedral or a an Ice Palace with long-decaying tails of sounds. As with audio signals, the discrete-component model for multipath is typical of outdoor propagation, especially in rural and suburban areas. It still holds in urban areas, with a more crowded delay spectrum. An example of the delay spectrum of outdoor urban propagation is given in Fig. 4.4 (a). Continuous-time "reverberation" is on the contrary typical of indoor propagation, where the delays are much smaller than outdoors, and the echoes tend to be perceived by the receiver as superimposed. See for instance the power profile of an indoor channel represented in Fig. 4.4 (b) and compare it with (a): the propagation paths ar many more, and the delays are much smaller. The "peaks" in the responses can be identified as the "rays" of multipath propagation, and a model based on concentrated echoes (deltas in the impulse response) as the one in Fig. 4.3 can be extracted.

4.1.2 Frequency- and time-selective channel

After the basics about multipath propagation are cleared, we are now to try to understand to what amount the multipath component in (4.4) is detrimental to data detection. Is it always true that multipath is detrimental indeed, irrespective of the parameters of the channel and of the signal? The question has a purely rhetorical intent. Just to take an extreme attitude, assume we do an indoor digital link at 1 ksymbol/s, thus with T=1 ms. Indoor multipath is characterized by (Fig. 4.4 (b)) delays at most equal to 100 ns, four order of magnitudes

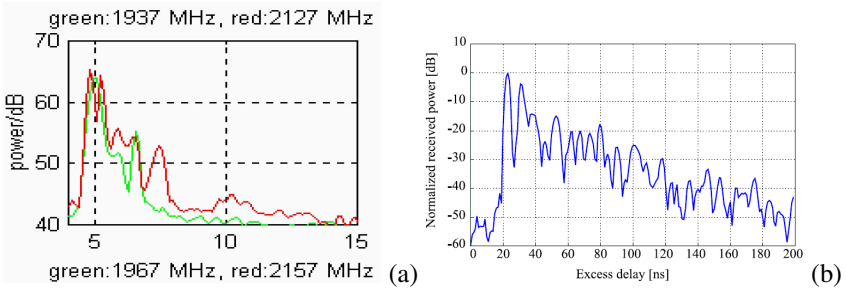


Figure 4.4 Measured Power profile of typical multipath channels. (a) Outdoor urban, and (b) indoor.

smaller than the symbol time T. It is quite apparent that, on the time scale of the symbol time, all echos are "heard" by the receiver as a *single* one, with no resolvable difference in delay among them. There's no "echo" whatsoever. Generalizing a little bit, we may say that whenever $\tau_i \ll T \ \forall i$, then, for the purpose of modeling the signal within a symbol-time interval, we can neglect all τ_i's in (4.2) and we have:

$$x(t) \cong s(t) \sum_{i=1}^{N_R} a_i \exp(\jmath\phi_i) = A \exp(\jmath\Phi) \cdot s(t) \tag{4.6}$$

where

$$A \exp(\jmath\Phi) \stackrel{\triangle}{=} \sum_{i=1}^{N_R} a_i \exp(\jmath\phi_i) \tag{4.7}$$

is a complex-valued constant that accounts for an overall amplitude and phase shift experienced by the signal. There is no *distortion* on the signal, nor additional components (because there is no multipath, actually). If we take the presence of the channel noise back into consideration, we see that we fall back onto an AWGN channel mode. The receiver has to estimate the value of the signal amplitude and phase (for coherent detection) as usual.

This is not true when on the contrary we have $\tau_i \approx T$ for some i. In that case the reasoning above fails, and we have to look further into the issue. Since we already characterized the input-output relation of the multipath channel as that of an LTI system, it makes sense trying to derive the *frequency response* $H_C(f)$ of that system. In general we get immediately from (4.5)

$$H_C(f) \stackrel{\triangle}{=} \mathcal{F}[h_C(t)] = \sum_{i=1}^{N_R} a_i \exp\{-\jmath(2\pi f \tau_i - \phi_i)\} \tag{4.8}$$

This expression is a little bit cumbersome and does not add much as it stands to the problem. It does indeed clarify a lot of things if we restrict ourselves to the simpler case with $N_R = 2$ and with the normalization (4.3). In that case, we get the response of the so-called *two-ray channel* (also called *Rummler's model*) which was the standard benchmark to evaluate the performance of anti-multipath countermeasures in good-old microwave radio equipment for backbone wireless communications. Its frequency response is

$$H_2(f) = 1 + a_2 \exp\{-\jmath 2\pi f \tau_2 + \jmath\phi_2\} = 1 - b \exp\{-\jmath 2\pi (f - f_N)\tau\} \tag{4.9}$$

where we adopted the standard notation that is used for such simplified channel model, namely, $\tau = \tau_2$ and $b = a_2$ just for simplicity, while f_N, that is called the *notch frequency*,

is such that $2\pi f_N \tau = \phi_2 + \pi$ so that the "minus" sign between the two components pops out. Consider now the shape of the amplitude response

$$|H_2(f)| = \sqrt{1 + b^2 - 2b\cos[2\pi(f - f_N)\tau]} \tag{4.10}$$

that is plotted for a few values of b in Fig. 4.5. It is seen that the response is *periodic* in the frequency domain with period $1/\tau$. Within the period, the response has one maximum equal to $|1 + b|$ and a minimum $|1 - b|$ that is attained for $f = f_N$. If $b = 1$, i.e., if the amplitude of the "secondary" path with delay τ is equal to the amplitude of the main path, then the response at $f = f_N$ is equal to 0. The curve has a 'deep valley" for that frequency, that is called the *notch* frequency. The notch is still there even if $b < 1$, although less deepened towards zero. The shape of this response raises some concern. We may well understand that a considerably wideband signal that encounters this two-ray channel experiences a selective modification of the components of its spectrum, according to $X(f) = S(f) \cdot H_C(f)$. What do we mean with "considerably" wideband? From Fig. 4.5 it is seen that if the bandwidth B of the input signal $s(t)$ is much smaller that the repetition period $1/\tau$ of the channel response, then such spectrum will *not* be treated selectively, as is apparent from a glance to Fig. 4.6 (a). This is the situation when $B \ll 1/\tau$, or, for a data signal, $1/T \ll 1/\tau$ that is, $\tau \ll T$. The signal on a carrier f_0 will experience a "flat" channel across its bandwidth with no distortion, but with an amplitude/phase shift: $x(t) = H_C(f_0)s(t)$. This is exactly what we already discovered in (4.6) with our time-domain analysis of the general multipath channel. The coefficient $A\exp(j\Phi)$ there, is just $H_C(f_0)$ here, and we are in a situation of *frequency-flat* fading with reference to the flat frequency response "seen" by the signal.

The frequency-flat condition can also be formulated in terms of the *coherence bandwidth* B_{coh} of the channel. This parameter represents the frequency span over which the amplitude response of the channel is substantially invariant and, in our two-ray example, amounts to a small fraction of the (frequency-domain) repetition period of the channel response τ:

$$B_{coh} \triangleq \frac{1}{10 \div 100}\frac{1}{\tau} \tag{4.11}$$

. If $B_{coh} > B \approx 1/T$, we may say that the fading is flat and all components of the signal spectrum are faded "in parallel", all with the same amplitude/phase channel response. If on the contrary we have $B \approx 1/\tau$, that is, $\tau \approx T$ then the spectrum of the data signal occupies a relevant fraction of the frequency repetition period of the channel response as in Fig. 4.6 (b), and the spectral components will experience a selective behavior. The signal $x(t)$ is a distorted version of $s(t)$, and the propagation condition is called *frequency-selective*: the multipath-induced term in (4.4) can be very annoying, and countermeasures in the receiver have to be adopted. The diverse techniques for multipath estimation/compensation will be discussed in detail in the next Chapter, according to the kind of signal they have to be applied to.

Example 4.35

Assume we have an ideal NRZ-BPSK wireless link on a two-ray multipath channel with a mild-to-severe multipath condition described by $a_2 = 0.5$, $\phi_2 = \pi$ and $\tau_2 = T/2$. In two-ray's notation this is equivalent to $b = 0.5$, $\tau = T/2$, and $f_N = 0$. Let us evaluate the (degraded) performance of the matched filter receiver assuming multipath + AWGN and making the further assumption that the receiver can synchronize to the direct ray. Skipping some details left to the reader, the symbol-time sampled output of the matched filter at time kT is

$$r[k] = As[k] - 0.5As[k] + n[k] = 0.5As[k] + n[k] \tag{4.12}$$

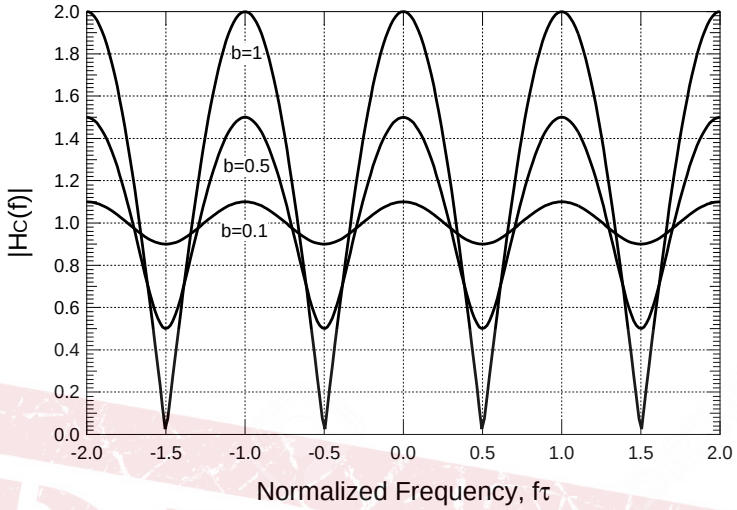


Figure 4.5 Amplitude response of the two-ray channel with $f_N = 1/(2\tau)$.

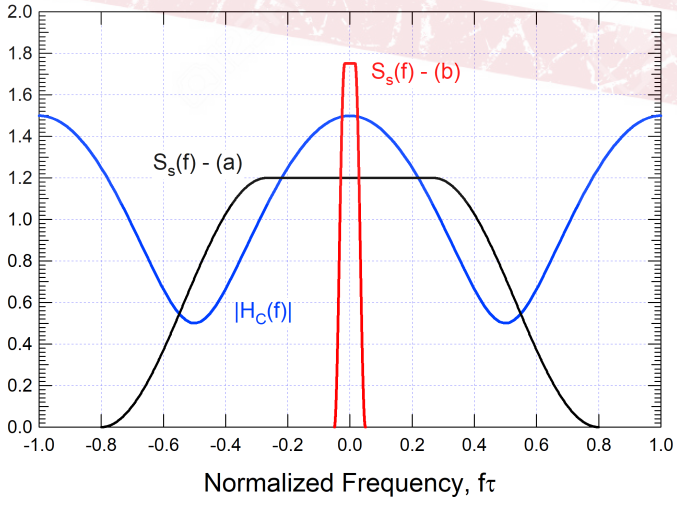


Figure 4.6 Frequency-selective (a), and Frequency-flat (b) propagation.

We see that the amplitude of the useful component is reduced by a factor 2 (the power is reduced by a factor 4) with respect to the case $b = 0$ of no multipath. So the matched filter has a degradation in terms of SNR equal to 6 dB.

When the multipath is "rich", i.e., it has a number of components larger than the two that we have considered above as in the general description (4.4), the definition of coherence bandwidth still applies, provided that, instead of tau in (4.11) we use the value of the maximum (differential) delay in (4.4), τ_{N_R}.

4.1.3 Fading channels

Until now, we have implicitly assumed that the two ends of the radio link were located at fixed points. But the most fantastic feature of radio terminals for wireless communication is that they can move around while still keeping the link alive ! Our description (4.2) of the multipath channel is to be intended as a *snapshot* of what is actually happening when movement of one of the link terminals (or both) comes into play. To understand what is happening in this case, it is expedient to consider two separate *time scales*: on one side we have the "fast" time of data symbols, with a rate of at least some tens of kBaud; on the other we have the slower time scale that describes the changes in the environment due to the motion of the radio terminal(s). That is what we mean with "snapshot": the time still appears in (4.2), but it is the fast time of data. To build a more accurate model, we have to re-introduce the slow time of terminal motion. And we have to introduce it to describe the changes that are usually experienced in slow motion with respect to the symbol rate in the *amplitude*, *phases*, and *delays* of the multipath components. The result is an updated multipath description as follows:

$$x(t) = \sum_{i=1}^{N_R} a_i(t) \exp(\jmath\phi_i(t)) \cdot s(t - \tau_i(t)) = \sum_{i=1}^{N_R} h_i(t) \cdot s(t - \tau_i(t)) \qquad (4.13)$$

Now the two time scales are mixed up, and it's up to us to tell them apart through an appropriate characterization of the kind of variability in time of the multipath parameters.

How can we do this? Is the time evolution of, say, $a_i(t)$ predictable? Again, we could theoretically predict the waveform of all the multipath parameters if we had accurate information on the environment, on the irradiation characteristics of the antennas, and of the trajectory followed by the mobile(s). That of course should be different case by case, according to the environment, the movement, etc. This sound like *mission impossible*. A reasonable approach to devise a useful model for design and analysis of communication equipment is seeking for a *statistical* model based on a few simplified parameters of the motion and of the environment: the speed of a car, urban vs. rural propagation and so on. Such model will fit a specific case in a statistical sense only, but will be what we need to do good design and analysis. Of course, statistical modeling can only be based on real-world measurements followed by thorough statistical analysis and fitting.

The main findings of such extensive activities are relatively simple. For the purpose of design and analysis of a communication system, is is found that the time variation of $\tau_i(t)$ can be safely neglected in a first approximation. The change in the group delay of the narrowband components of the modulated signals takes place on a much longer time scale

than T, so that we can assume $\tau_i(t) \equiv \tau_i$, and our multipath model reads

$$x(t) = \sum_{i=1}^{N_R} [h_{i,I}(t) + \jmath h_{i,Q}(t)] \cdot s(t - \tau_i) \tag{4.14}$$

On the contrary, the amplitude and phase variation of the wave carrier at frequency f_0 is relatively fast and has to be taken into account. Is there statistical regularity in their evolution?

Before answering this general question, let us take back into consideration the simpler issue of the frequency-flat, invariant channel, wherein $x(t) = A\exp(\jmath\Phi)s(t)$. If we add to this the relative motion of the radio terminal, we get

$$x(t) = A(t)\exp(\jmath\Phi(t))s(t) = \psi(t) \cdot s(t) \tag{4.15}$$

where $\psi(t) = \psi_I(t) + \jmath\psi_Q(t)$ is complex-valued. It happens that what is measured at the receiver as a single component is actually produced by the superposition of many smaller components with tiny differential delays that combine at the receiver as a single one. The result of such composition has *Gaussian statistics* so that, to a good approximation, the I and Q components of $\psi(t)$, that is, $\psi_I(t)$ and $\psi_Q(t)$ are mutually independent WSS Gaussian processes with statistical characteristic to be determined. In a typical urban environment, the two components are zero-mean and have equal variance σ_ψ^2. The power of the fading process is thus $P_\psi = 2\sigma_\psi^2$. To keep the same received power as we get out of the AWGN channel, we may normalize $\psi(t)$ so that $\sigma_\psi^2 = 1/2$. A typical realization of the process is shown in Fig. 4.7 in the form of amplitude $A(t)$ and phase $\Phi(t)$. The amplitude occasionally dives down deep towards zero, and the radio signal in those time instants appears to *fade*. The "complex envelope" $\psi(t)$ of the signal in a mobile radio link is therefore called the *fading process*, and the communications channel is called the *flat-fading* channel.

As already mentioned, within a typical urban environment the fading process is zero-mean. Its normalized instantaneous amplitude $A(t) = [\psi_I^2(t) + \psi_Q^2(t)]^{1/2}$ has a Rayleigh pdf

$$f_A(a) = \frac{2a}{P_\psi} \cdot \exp\left(-\frac{a^2}{P_\psi}\right) \quad , \quad a > 0 \tag{4.16}$$

whilst the instantaneous phase $\Phi(t)$ has a uniform distribution on $[0, 2\pi)$. This is why the flat fading in an urban environment is also called *Rayleigh fading*. In a less densely built area, it is likely that there always exists a direct line of sight between the transmitter and the receiver, so that the randomly-varying term is accompanied by a fixed, non-random component, and the fading process is no longer zero-mean: $E\{\psi(t)\} = \psi_0$, where ψ_0 is just the line-of-sight component. In this case the total power of fading is $P_\psi = |\psi_0|^2 + 2\sigma_\psi^2$ (that we will assume equal to 1 for normalization), and the statistical law for the total fading amplitude follows *Rice*'s distribution

$$\begin{aligned} f_A(a) \quad &= \quad \frac{2(K_r+1)}{P_\psi} a \exp\left[-\frac{(K_r+1)a^2 + K_r}{P_\psi}\right] \\ &\cdot \quad I_0\left(2\sqrt{\frac{K_r(K_r+1)}{P_\psi}}\, a\right) \quad , \quad a > 0 \quad , \quad K_r \triangleq \frac{|\psi_0|^2}{2\sigma_\psi^2} \end{aligned} \tag{4.17}$$

where the *Rice factor* K_r is the ratio between the power received through the LOS path $|\psi_0|^2$ and the total power received from the fading components, $2\sigma_\psi^2$, and where $I_0(\cdot)$ is the modified zero-order Bessel function of the first kind. We have in this case the *Rice fading* channel.

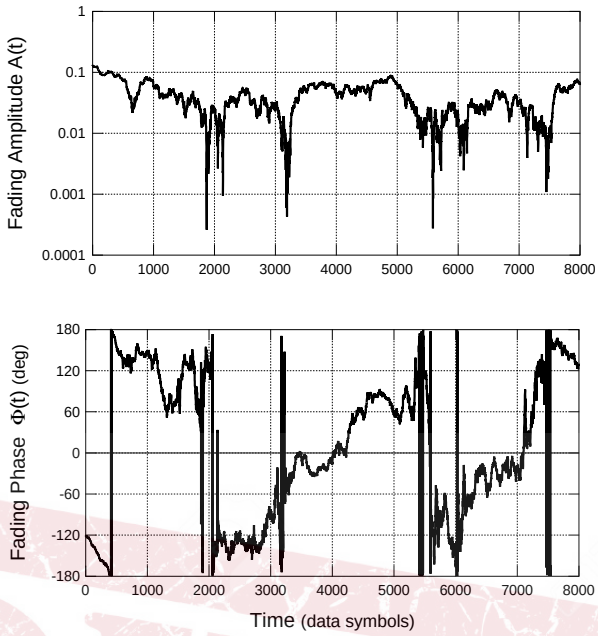


Figure 4.7 Sample Amplitude and Phase fading processes

Example 4.36

We take back into consideration the bread-and-butter case of BPSK with NRZ pulses. What happens to the BER of the matched filter if the signal propagates through a Rayleigh flat-fading channel and is corrupted by AWGN? To answer this question, we add the hypothesis that the fading process varies slowly with respect to the symbol time. On the signaling interval $0 \leq t < T$ then

$$x(t) = \psi(t)s(t) \approx \psi(0) \cdot s(t) \quad \Rightarrow \quad r(t) = \psi(0) \cdot s(t) + w(t) \tag{4.18}$$

where $w(t)$ is AWGN and $\psi(t)$ is normalized so that $E\{|\psi(t)|^2\} = 1$. To recover the BPSK data, the phase of the fading channel has to be compensated for. For simplicity, we will assume that such compensation is ideal, so that our final signal model is

$$z(t) = A \cdot s(t) + w(t) \tag{4.19}$$

where $A = |\psi(0)|$ is a unit-power Rayleigh random variable. Without entering all details, it is apparent that if the value of the fading amplitude A were known, the *conditional* BER of the receiver would be

$$BER|A = Q\left(\sqrt{\frac{2A^2E_b}{N_0}}\right) = \frac{1}{\sqrt{2\pi}} \int_{A\sqrt{2E_b/N_0}}^{\infty} \exp(-\beta^2/2)d\beta \tag{4.20}$$

where E_b/N_0 is the SNR ratio on the un-faded AWGN channel ($A \equiv 1$). The overall BER considering the fading channel is just the *average* of (4.20) on the Rayleigh statistics of

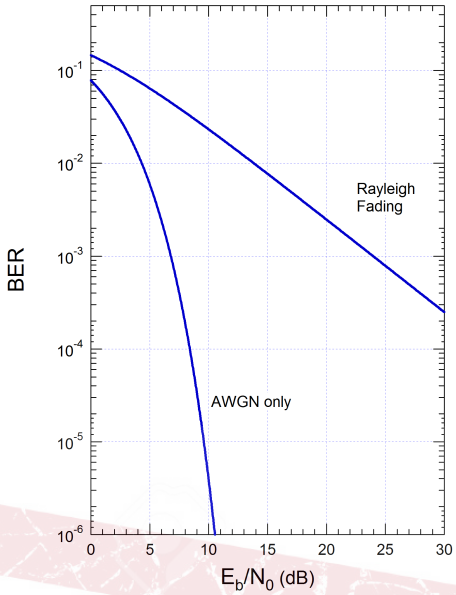


Figure 4.8 Comparison between the BER on the AWGN and on the Rayleigh channels.

fading:

$$BER = E_A\left\{Q\left(A\sqrt{\frac{2E_b}{N_0}}\right)\right\} =$$

$$BER = \int_0^\infty 2\alpha\exp(-\alpha^2)\cdot Q\left(\alpha\sqrt{\frac{2E_b}{N_0}}\right)d\alpha \tag{4.21}$$

This integral can be solved by parts (the finite factor is the BER value). The final result is

$$BER = \frac{1}{2}\left(1-\sqrt{\frac{E_b/N_0}{1+E_b/N_0}}\right) \tag{4.22}$$

The difference between the BER on the AWGN channel and the channel with flat Rayleigh fading as shown in Fig. 4.8 is striking. The increase of the BER is caused by those time interval that see a deep fade in the amplitude of the received signal with a very poor "local" SNR.

Still, we have to say more about the kind of time-variability of the fading process, in particular *how fast* the fading components $\psi_I(t)$ and $\psi_Q(t)$ change in time. From elementary physics, it is known that the instantaneous phase of a carrier wave received from a moving transmitter changes in time producing a shift of the received instantaneous frequency known as *Doppler shift*. The Doppler shift Δf is simple to compute for the case of a terminal that moves at constant speed v on a straight line, as shown in Fig. 4.9:

$$\Delta f = \frac{v}{c}f_0\cdot\cos(\gamma) \tag{4.23}$$

where c is the speed of light and γ is the angle between the direction of the motion and the line that joins the transmitter and the receiver. Even if the speed is constant, the Doppler shift changes due to the change of the angle γ, and this is the well-known modulation of the frequency of a siren of an ambulance that speeds to the rescue. Since we are assuming a narrowband signal, the model of the received signal with Doppler shift is

$$x(t) = As(t)e^{j(2\pi\Delta ft+\Phi)} \tag{4.24}$$

where A and Φ are the amplitude and phase of the carrier. In a composite environment like the one that generates the flat fading, we know that we have a number of different radio components coming from different angles γ_i to the receiver. This means that we have "many" Doppler shifts $\Delta f_i = (vf_0/c) \cdot cos(\gamma_i)$ according to the many values of the angle γ_i corresponding to the randomly-distributed *scatterers* in a certain environment, so that the received signal is now

$$x(t) = s(t)\left[A_1 e^{j(2\pi\Delta f_1 t+\Phi_1)} + A_2 e^{j(2\pi\Delta f_2 t+\Phi_2)} + \dots\right] = s(t) \cdot \psi(t) \tag{4.25}$$

It is seen that the components received from the environment generate scattered frequencies around the carrier frequency f_0. When such components are very many, the discrete spectrum of $\psi(t)$ becomes *continuous* with respect to the frequency, with infinite components. The bandwidth of this spectrum $S_\psi(f)$ (baseband equivalent) is the *maximum Doppler shift* that we may ever experience ($\gamma = 0$):

$$f_D = \frac{v}{c} f_0 \tag{4.26}$$

This is called the *Doppler spread* because it gives the amount of spreading of the frequencies of the different components that are received in a fading channel due to scattering of the radio waves.

The actual *shape* of the spectrum (or, to be more accurate, of the psd $S_\psi(f)$ of the fading process) within this bandwidth is difficult to say, since it depends on the particular distribution of scatterers in the environment (buildings, trees etc.) and on the actual trajectory of the mobile (speed, acceleration etc.). A reasonable assumption is that the psd $S_\psi(f)$ is flat within the bandwidth $B = f_D$, with a total power $P_\psi = 1$. Other models are used in the practice, but they are somewhat arbitrary (for instance, Jakes' model with a popular U-shaped psd) since they refer to a specific assumption on system parameters (for Jakes' psd, uniform speed on a straight line and uniform distribution of scatterers on the ground).

Assuming that we now know the psd of the fading process, we also have a fairly good idea about its kind of time variability. To make an example, take a car on a German highway speeding at v =144 km/h (or 40 m/s). The Doppler spread on a carrier at the frequency of UMTS downlink f_0 =2.14 GHz is f_D =285.3 Hz. This means that the fading process we experience changes as fast as 285 times per second. Is it fast or not? To answer, we have to compare this figure with the signalling rate $R = 1/T$. As a rule of thumb, if $f_D \leq R/10$, then we may say that our fading is *slow*, and the assumption we made in our Example 36 is verified. The fading process is constant when observed on the symbol time T, and there is no *time selectivity*. We may also formulate this statement saying that the *coherence time* of the channel T_{coh} (i.e., the time span over which the channel is substantially invariant) is longer than the symbol interval. If we are not in such condition (i.e., we have $T_{coh} \approx T$), we have *fast fading* or *time-selective* fading, and a computation like the one in Example 36 is no longer valid, since the fading process change even on the short time scale of a data

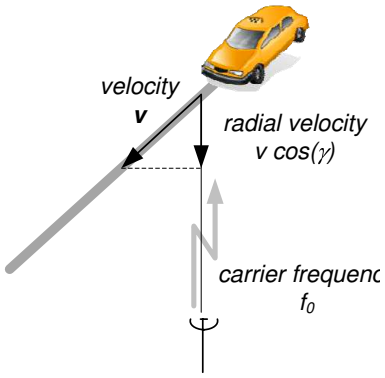


Figure 4.9 Computation of the Doppler shift Δf

symbol. An expedient definition for the coherence time of the channel, that resembles very much the definition of coherence bandwidth, is

$$T_{coh} = \frac{1}{10 \div 100}\frac{1}{f_D} = \frac{1}{10 \div 100}\frac{c}{v \cdot f_0} \tag{4.27}$$

Example 4.37

The probability of error on the flat Rayleigh-fading channel (4.22) has to be considered as a *long-term* average of a *short-term* BER given by (4.20). What we mean by this is that the data link we are considering is experiencing a BER that is variable with time. At times, it may be satisfactory for a certain application (say, low quality video), at times it may be not. This depends on the particular value of the Rayleigh-fading amplitude in (4.20). So, instead of the long term, overall BER caused by fading, we may interested in the *fraction of time* within which the BER is satisfactory for our application, that is, BER $\leq$ BER$_0$, where BER$_0$ is the application-dependent target value or *Quality-of-Service* (QoS). We may assume that BER$_0$ is the value we would get at the output of the un-faded AWGN channel ($A = 1$). In this case, the probability that the link is "satisfactory" is

$$\Pr\{\text{BER} \leq \text{BER}_0\} = \Pr\{A \geq 1\} = \int_1^\infty 2\alpha \exp(-\alpha^2) d\alpha = 1/e \tag{4.28}$$

This is called the *availability* of the link, and its complement $1 - 1/e$ is called the *outage probability*. This outage probability is not satisfactory (it is roughly equal to 63%), so that the designer is forced to introduce a *margin*: he/she has to require that the (nominal) value of E_s/N_0 at the output of the un-faded channel, which is also equal to the *average* value of E_s/N_0 at the output of the faded channel, be larger by a factor M (usually expressed as $10 \log M$ dB) than the bare minimum we considered before. In that case, the outage probability is

$$\Pr\{\text{BER} \geq \text{BER}_0\} = \Pr\left\{A\sqrt{M} \leq 1\right\}$$

$$= \int_0^{1/\sqrt{M}} 2\alpha \exp(-\alpha^2) d\alpha = 1 - \exp(-1/M) \tag{4.29}$$

The outage probability shown in Fig. 4.10 as a function of the margin, can be made arbitrarily small to meet a prescribed QoS requirement, at the expense of additional transmitted power.

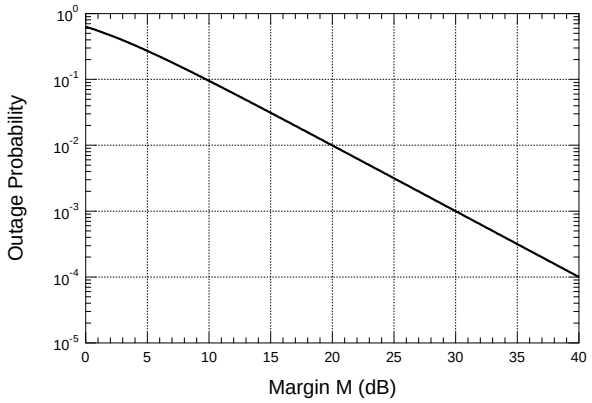


Figure 4.10 Outage probability of the Rayleigh flat-fading channel

Now that the description of the case of flat fading is complete (amplitude statistics and spectral characteristics), we come back to the issue of frequency-selective fading. When we have relative motion of transmitter and receiver, the analysis that we performed on the flat-fading process has to be applied to *any* of the different multipath components of (4.14). We have multiple Rayleigh fading on each component with the same Doppler spread as in the flat-fading case, so that each $h_i(t)$ in (4.13) has the same statistical characterization as $\psi(t)$ in (4.15). We may possibly have Ricean fading on the first path if we also have a LOS connection. And we speak again of slow or fast fading according to the value of f_D/R. From this discussion, we easily understand that on the multipath channel we may have selectivity in *both* frequency and time. This is of course the most challenging propagation condition, but fortunately it is seldom verified in the practice.

4.1.4 Channel modeling

The description that we gave for static multipath in the form of a delay spectrum/power profile plot as in Fig. 4.4 is valid for the time-varying channel as well. Assume that we intend to build a C++ time-domain simulator of a multipath time-varying channel. We already have a set of samples of the data-modulated signal $s(t)$ (baseband equivalent, of course), and we are to produce the time-domain samples of $x(t)$ as in (4.14). We have to know first the carrier frequency and motion speed, so that we can derive the Doppler spread. Then we have to simulate the fading processes $h_i(t)$ on each path. To do so, we can to start from N_R independent white Gaussian processes that are easily simulated with random number generators. Then we have to filter them with a bank of low-pass filters with bandwidth f_D each, and then we have to scale them so that each have a certain *power* P_{h_i} as specified in the power profile plot. We see that no information on the phase of the i-th component is needed, only amplitude information is required. Then we have to process the samples of $s(t)$ to provide the delayed samples with the delays given by the delay spectrum plot. Either we approximate such delays with multiples of the time quantum in our simulation, or we do some kind of interpolation of the available samples to regenerate the delayed ones. And finally we build the linear combination of the delayed samples with coefficients given by the random processes as in (4.14). Everything that we need is the

i	ρ_i	τ_i [μs]	θ_i [rad]
1	0,057 662	1,003 019	4,855 121
2	0,176 809	5,422 091	3,419 109
3	0,407 163	0,518 650	5,864 470
4	0,303 585	2,751 772	2,215 894
5	0,258 782	0,602 895	3,758 058
6	0,061 831	1,016 585	5,430 202
7	0,150 340	0,143 556	3,952 093
8	0,051 534	0,153 832	1,093 586
9	0,185 074	3,324 866	5,775 198
10	0,400 967	1,935 570	0,154 459
11	0,295 723	0,429 948	5,928 383
12	0,350 825	3,228 872	3,053 023
13	0,262 909	0,848 831	0,628 578
14	0,225 894	0,073 883	2,128 544
15	0,170 996	0,203 952	1,099 463
16	0,149 723	0,194 207	3,462 951
17	0,240 140	0,924 450	3,664 773
18	0,116 587	1,381 320	2,833 799
19	0,221 155	0,640 512	3,334 290
20	0,259 730	1,368 671	0,393 889

Table 4.1 DVB-T standard multipath channel

power/delay profile, plus the Doppler spread. This is why we said earlier than the phase spectrum is of no avail. We need information on the phases of the paths only if we want to build a *deterministic* model of a time-invariant multipath channel. For instance, the DVB-T standard for digital terrestrial television specifies a reference static multipath channel as in Tab. 4.1. The power/delay profile also differentiates the different kind of environment that may be encountered. We already highlighted in Fig. 4.4 the difference between outdoor and indoor propagation, the first one being characterized by a small number of paths and delays of the order of 1 to 10 μs, whilst the second one with tens of rays, and delays ranging from 10 to 100 ns. Table 4.2 shows the specifications of the statistical model for the pedestrian/vehicular/urban channels according to the 3G/4G/5G standardization body `www.3gpp.org` an urban channel (a) and a hilly terrain channel (b). In the urban environment, the dominating path is not the shortest one (that may well be blocked by a large building), and many path with small delays are experienced. In the pedestrian model delays are very small, representing the short distances ofn urban canyon/indoor environment.

Very often, a slightly different approach than the *parametric* model (4.2) for channel modeling is adopted. The impulse response $h_C(t)$ of the multipath channel based on delta functions representing the delays of the different paths as in (4.5) cannot be actually observed or measured since a delta function ha infinite bandwidth. What we get when we perform "channel sounding", that is, measurement of the channel impulse response, is actually a *bandlimited* version of $h_C(t)$ that looks very much as in Fig. 4.4. The discrete-time components in $h_C(t)$ are actually "smeared" into a continuous-time, bandlimited response. This is not a source of inaccuracy in modeling or measurements, since any radio signal that is used in communications is bandlimited. The only precaution that we have to adopt is that the measurement bandwidth of the response be larger that that of the signal that will be actually used on the channel. As a consequence, an equivalent model to describe the behavior of the "smeared", bandlimited response of the channel is the *tapped delay-line* model, wherein the amplitude (power) of the channel response is listed in a table or represented in a chart with a regular "sampling" step of the delay that represent regular sampling of the bandlimited response. If the radio signal $s(t)$ is bandlimited as well, the

Extended Pedestrian A model (EPA)

Tap	Excess tap delay [ns]	Relative power [dB]
1	0	0.0
2	30	-1.0
3	70	-2.0
4	90	-3.0
5	110	-8.0
6	190	-17.2
7	410	-20.8

Extended Vehicular A model (EVA)

Tap	Excess tap delay [ns]	Relative power [dB]
1	0	0.0
2	30	-1.5
3	150	-1.4
4	310	-3.6
5	370	-0.6
6	710	-9.1
7	1090	-7.0
8	1730	-12.0
9	2510	-16.9

Extended Typical Urban model (ETU)

Tap	Excess tap delay [ns]	Relative power [dB]
1	0	-1.0
2	50	-1.0
3	120	-1.0
4	200	-0.0
5	230	-0.0
6	500	-0.0
7	1600	-3.0
8	2300	-5.0
9	5000	-7.0

Table 4.2 3gpp standard multipath channels

result of the (discrete-time) convolution with that response will be

$$x(t) = s(t) \otimes h_C(t) = \sum_{i=0}^{N_{tap}-1} h_i s(t - iT_s) \tag{4.30}$$

where T_s is the tap spacing (the unitary delay of the delay line). The delays in the delay line do not correspond to delays of the physical paths any longer, as well as the values of the (complex-valued) taps h_i lose the physical meaning of actual amplitude and phases of the propagation paths. Nonetheless, the tapped delay-line is as accurate as the discrete-component model we have considered until now.

4.1.5 Large-Scale Radio Propagation Modeling

The channel models that we have discussed and used until now are also called *small-scale models* since they describe in great detail what happens on a single point-to-point link in the time domain and in the frequency domain, but fail to capture the large variations that may be experienced on a larger scale in the wide coverage area of a whole cellular network. For instance, assuming free-space propagation, we know that the signal power P_R at the receiver can be evaluated as

$$P_R = P_T \cdot G_T G_R \left(\frac{\lambda_0}{4\pi d}\right)^2 \triangleq P_T \cdot L(d) \tag{4.31}$$

where $L(d)$ is the so-called *path loss* that relates the received power with the transmitted one. This relation shows the dependence of P_R on HW parameters like the receive and transmit antenna gain, but also (through the path loss) on the large-scale parameter d, i.e., the TX-RX distance, that is not present in any of the small-scale models of the previous sections, wherein the total transmitted power is a given. A different kind of description is clearly needed, and it is also apparent that a simple relation like (4.31) is not enough as long as the propagation scenario is richer than the simple free-space - pretty much always !

4.1.5.1 Large-Scale Variation of the Received Power when d is (approximately) constant

Let us first assume that we have a link between a BTS and a MT that is moving along a circle centered at the BTS, as in Fig. 4.11, within a typical urban scenario (also called *Manhattan* urban model for self-evident reasons). Since we are not in free-space, on top of the general average received power predicted by , and regardless of the finer, short-term details of multipath that we have already considered, we will find a long(er)-term fluctuation of the signal strength that is not taken into account by the Doppler-spread modeling of the previous sections, just because it is caused by large-scale variation of the propagation scenarios (in and out of an alley, different characteristics of the building on the radio path etc.). Specifically, the (average) received power $P_R(d)$ at distance d is a random variable and follows a log-normal distributions, i.e., its value expressed in dB

$$\Gamma(d) \triangleq 10 \log_{10} P_R(d) \tag{4.32}$$

is a Gaussian variable with pdf

$$f_\Gamma(\gamma) = \frac{1}{\sqrt{2\pi\sigma_\Gamma^2}} \exp\left(-\frac{(\gamma - \overline{\Gamma}(d))^2}{2\sigma_\Gamma^2}\right) \tag{4.33}$$

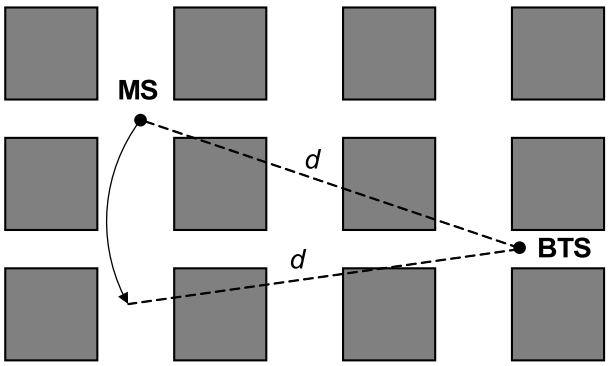


Figure 4.11 Circular motion in an Urban Scenario

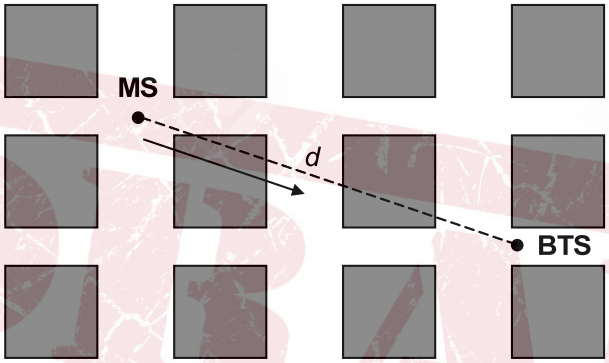


Figure 4.12 Radial motion in an Urban Scenario

where the average received power $\overline{\Gamma}(d)$ is a function of the distance and of the cell scenario, whilst the variance σ_{Γ}^2 is only a function of the specific scenario. This log-normal distribution ia sometimes called *large-scale fading* and can be taken into account when considering the different propagation conditions that arise in a cell because of the different scenarios in that cell.

4.1.5.2 Large-Scale Variation of the Received Power when d is variable Let us focus now on the dual case wherein the MS is (ideally) moving on a straight line directly toward the BTS, i.e., radial motion as in Fig. 4.12). The distance d is now variable, so that the average received power is not constant any longer - in other words, we have to find the law of variation of $P_R(d)$ (or $\overline{\Gamma}(d)$)), taking also into account the variability of the call scenarios. After well-consolidated experimental measurements, a well accepted analytical model to predict the average received power at distance d is Hata-Okomura's, that generalizes the free-space path loss equation:

$$P_R(d) = \begin{cases} P_T \cdot L(d) = \chi' \cdot P_T/d^2, & \text{if } d \le d_0 \\ \chi \cdot P_T/d^n = \left(\chi' \cdot d_0^{n-2}\right) P_T/d^n, & \text{if } d \ge d_0 \end{cases} \tag{4.34}$$

where $L(d)$ is defined in (4.31), so that $\chi' = G_T G_R[\lambda_0/(4\pi d)]^2$ and where d_0 is the so-called *reference distance* of H-O's model: on a shorter distance propagation is free-space

(no relevant influence of the propagation scenario), whilst on a larger range the signal is in general more attenuated than in free space. The exponent n is called *path loss exponent*, while χ is just a continuity factor at the boundary $d = d_0$:

$$\chi = \chi' \cdot d_0^{n-2} \tag{4.35}$$

The values of d_0 and especially n are derived, as already mentioned, after extensive measurement campaigns. In summary, the rule-of-the-thumb is as follows:

- $d_0 = 100\,\mathrm{m}$ for urban scenarios, $d_0 = 1\,\mathrm{km}$ for rual areas;
- $2.7 \leq n \leq 3.5$ up to more than 4 for especially dense urban areas, and down to 2.5 for the countryside.

The macro-modeling of the latter sections are the basis for the design of the cells layout of a cellular network, in particular of the interference calculations across cells (Signal-to-Interference Ratio SIR), and the evaluation of the signal to noise ratio for the users population. Once the cell layout is such that the design criteria for SNR and SIR are satisfied, then the finer, small-scale aspect of multipath propagation are taken into account.

CHAPTER 5

A RAINBOW OF DATA – MULTICARRIER TECHNOLOGIES

"Somewhere over the rainbow/Skies are blue/And the dreams that you dare to dream/Really do come true"

—1939, from the soundtrack of "The Wizard of Oz", Judy Garland singing

One of the dreams of the communications engineer is that of designing a good signaling scheme for transmission of high bit-rate signals over the wireless channel. That is, being robust enough to withstand the formidable distortion experienced by the radio signal during propagation into the multipath channel – just like building a signal that looks *wideband* and *narrowband* at the same time... The solution is *multicarrier modulation*: placing many *narrowband* data signals side by side within the (wide) transmission bandwidth, on adjacent (sub)carriers, and in such a way that each one no longer experiences distortion from the channel: a rainbow of data of different *colors* that make the dream come true.

5.1 Multicarrier Communications

5.1.1 Multipath and Channel Distortion

We already know that whenever the bandwidth B_x of a digital signal $x(t)$ transmitted onto a wireless channel turns out to be larger than the coherence bandwidth B_c of that channel, we encounter frequency-selectivity problems, and the received signal is in general distorted – the more $B_x > B_c$, the more severe the distortion is. The situation of frequency selectivity is synthetically represented in Fig. 5.1, where the frequency response of the wireless channels exhibits many "peaks and valleys" within the bandwidth of the digital signal. Unfortunately, as we have already know, as soon as the transmitted symbol rate exceeds some tens of kBauds, the bandwidth B_x will be larger than the coherence bandwidth B_c of any outdoor wireless scenario, and frequency selectivity starts kicking in. Current wireless waveforms, like Wi-Fi or LTE have a bandwdith occupancy B_x=20 MHz, way larger than the coherence bandwidth of any wireless channel, even in indoor scenarios with smaller delay spread. So, frequency selectivity becomes an issue, we would say *the* issue to be solved to build a robust digital radio link with any coding and modulation technology.

The conventional solution to the issue was invented back in the 60's for wireline telephone modems: introducing in the receiver a pre-filter, called *equalizer*, that tries to compensate for the channel distortion as shown in Fig. 5.2. The aim of the equalizer is to compensate, or at least mitigate, the frequency-selectivity of the transmission channel so that the overall (channel+equalizer) frequency response $H(f) \cdot E(f)$ be as flat as possible across the signal bandwidth. In addition, the propagation conditions of a wireless channel are typically time-variant (think of mobile communications) – the equalizer has therefore to be *adaptive* to track continuously the channel variations along time. The equalizer is usually implemented in the receiver as a linear Finite-Impulse Response (FIR) digital filter with sampling frequency equal to half the symbol time $T_s/2$ or less. When the channel exhibits a strong frequency-selectivity, the equalizer may be so complex (as we show in the following example) as to be one of the most critical sub-system of the receiver.

Example 5.38

Consider an $r = 1/2$ LDPC-coded single-carrier QPSK data link between a drone and its own base station, with a desired $R_b = 1$ Mbit/s. The signaling rate is $R_s = 1$ Mbaud, and the drone operates in a rural scenario which can be well modeled by a two-ray multipath channel with $\tau = 10\ \mu s = 10T_s$. The signal format is SRFRC with roll-off factor $\beta = 0.25$, and the demodulator at the base station is implemented through a conventional DSP-based architecture as in Fig. 2.32, where the ADC rate (sampling rate) is $f_{sa} = 1/T_{sa} = 2R_s$. The particular sampling rate gives *two samples per symbol* and obeys Nyquist rule for correct sampling of a bandlimited signal, since the (baseband) bandwidth of the I/Q components is $B = (1 + \beta)R_s/2 = 0.6125R_s$. The sampling rate

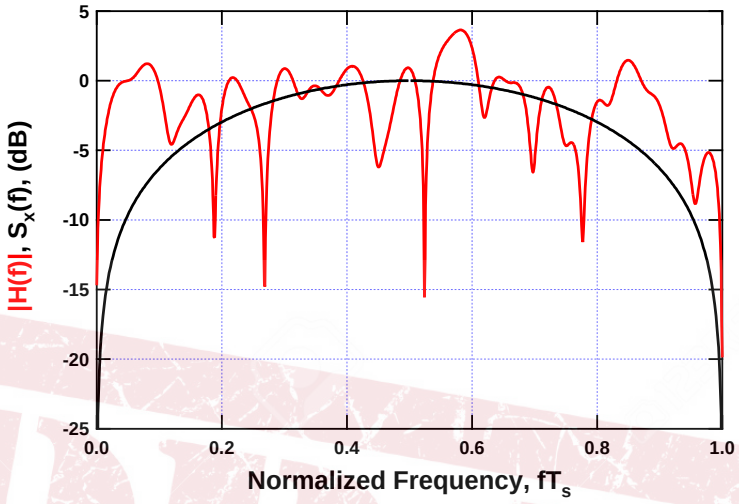


Figure 5.1 Representation of Frequency Selectivity

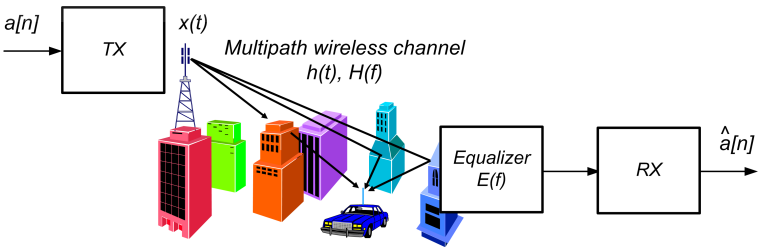


Figure 5.2 The Equalizer to compensate for Channel Distortion

in DSP-based modem is usually selected between 2 and 4 samples/symbol, and this figure is called the *oversampling factor* (as referred to the symbol rate).

Let us now try to design the equalizer to compensate for the distortion induced by multipath. From (4.9), the ideal frequency response of the equalizer $E(f)$ is

$$E(f) = \frac{1}{H_2(f)} = \frac{1}{1 - b\exp\{-\jmath 2\pi(f - f_N)\tau\}} \tag{5.1}$$

and we should find a way to design a digital filter with impulse response $e[n]$ that implements this filter in the digital domain. From (5.1), assuming $|b| < 1$, we get

$$E(f) = \sum_{\ell=0}^{\infty} (b\exp\{-\jmath 2\pi(f - f_N)\tau\})^{\ell} = \sum_{\ell=0}^{\infty} b^{\ell}\exp\{-\jmath 2\pi(f - f_N)\ell\tau\} \tag{5.2}$$

We know that the frequency response $H_2(f)$ of the two-ray channel has infinite bandwidth, so that the equalizer will be infinite-bandwidth as well. Its digital implementation will be a *bandlimited* version of $E(f)$, in a bandwidth that has to be no smaller than the signal bandwidth B not to introduce any aliasing error during digital filtering. We will just take the *full digital bandwidth* $f_{sa}/2 = R_s$ so that the bandlimited equalizer will be

$$E_B(f) = \text{rect}\left(\frac{f}{2R_s}\right)\sum_{\ell=0}^{\infty} b^{\ell}\exp\{-\jmath 2\pi(f - f_N)\ell\tau\} \tag{5.3}$$

or, taking the inverse FT to get into the time domain,

$$e_B(t) = 2R_s\sum_{\ell=0}^{\infty}(b\exp\{\jmath 2\pi f_N\tau\})^{\ell}\text{sinc}(2R_s(t - \ell\tau)) \tag{5.4}$$

The impulse response of the digital equalizer is obtained by the *impulse invariance* method, i.e., letting $e[n] = e_B(n/f_{sa})/f_{sa}$:

$$\begin{aligned} e[n] &= \sum_{\ell=0}^{\infty}(b\exp\{\jmath 2\pi f_N\tau\})^{\ell}\text{sinc}(n - 2R_s\ell\tau)) = \\ &= \sum_{\ell=0}^{\infty}\Psi^{\ell}\text{sinc}(n - 20\ell) = \sum_{\ell=0}^{\infty}\Psi^{\ell}\delta[n - 20\ell] \end{aligned} \tag{5.5}$$

where $\Psi \stackrel{\triangle}{=} b\exp\{\jmath 2\pi f_N\tau\}$ and where we have used the property of the sinc function $\text{sinc}(n) = \delta[n]$ (see Fig.2.6) and (2.68).

It is seen that the equalizer is an Infinite Impulse Response (IIR) filter, and its response is made of "isolated" samples spaced 20 samples apart (20 is the value of the multipath delay τ as measured in sampling times T_{sa}) – we call it a *sparse* signal. We may truncate this infinite response to turn it into that of an FIR filter, observing that the values of $e[n]$ different from zero decay exponentially since $|\Psi| = |b| < 1$. If the multipath is not too strong (i.e., $|b|$ not so close to 1), we can retain some 20 values and discard the subsequent ones. The truncated, FIR equalizer will have a total duration of 20*20+1=401 samples – very many! It is true that most samples are equal to 0, but this is just a very, *very* special case due to the particular value that we have taken for the multipath delay ($\tau = 10T_s = 20T_{sa}$), that turns out to be an *integer multiple* of the sampling time. When this condition is not true (almost always!), *all* of the 401 samples of the equalizer will be different from 0, so the filter becomes very much complicated to design and implement, especially when the equalizer has to be *trained* (to learn the optimum response (5.5)) on a preamble of the digital signal, and the filter has to be *adapted* to this situation – it may

take a long time, incompatible with the needs of fast setup of the digital link of our drone (or any other application, for that matter).

Example 38 taught us that the channel equalizer may need a number of coefficients (taps) on the order of a few hundreds. In addition, it has to be *adaptive* – as a trivial example, whenever we zap across TV channels, we have to let the equalizer re-adapt its own 400+ coefficients whenever we switch to a new channel (program), and this takes an intolerably long time.

The idea is to change the approach to frequency selectivity: instead of using a conventional digital format , and then try to mend with an equalizer the wrong doing of the channel, why not adopting form the onset a *more robust* digital format at the *transmitter* end? In particular, why not try to design a digital signal that, as the prologue to this chaptger suggests, looks at the same time narrowband (not to suffer channel selectivity) and wideband (to convey a high bit rate)? The answer to this question is: we have to use a *multicarrier* format.

In multicarrier modulation what we do is splitting the single data stream with high signaling speed $1/T_s$ (and therefore wide band) into a multitude of parallel sub-streams obtained by de-multiplexing the original flow, and then modulating with these many streams an equal number of adjacent *sub-carriers*. If we indicate with N the number of these sub-streams, the signaling rate on each subcarrier $R_M = 1/T_M$ (where $_M$ stands for Multicarrier) will be *slower* by a factor N than the original rate, $R_M = R_s/N$, and the symbol interval on each subcarrier will be $T_M = NT_s$, i.e., N times *longer*. Considering a generic sub-stream on a generic sub-carrier, the modulated signal band will be of the order of R_M, therefore N times smaller than the band $B_x = 1/T_s$ of the original single-carrier signal. In such a way, by increasing N appropriately, we can attain a situation wherein each sub-channel has a very narrow band, *narrower than the coherence bandwidth of the channel* B_c, so that the the sub-channel "sees" a slice of the frequency response of the channel that is substantially flat and distortionless. This situation is sketched in Fig. 5.3 where the colored spectra represent the different signals on the different, adjacent subcarriers. To make an example, in 4G LTE cellular communications the number of subcarriers is N=2048, and in the DVB-T2 standard for digital terrestrial television we can have as many as N=32768 subcarriers!

Looking a bit deeper into the issue, the only effect of the transmission channel on the generic sub-channel is to introduce an attenuation and phase shift factor, as happens on a frequency-flat channel – with the difference that this factor in general changes form subcarrier to subcarrier (i.e., from frequency to frequency) as is apparent form Fig. 5.3. Recovering these sub-carrier phase/amplitude factor for all of the different subcarriers in the spectrum of the signal is similar to a sort of "frequency equalization" of the received signal, in the sense that the receiver is automatically going to compensate for the frequency response of the wireless channel, component by component, across a very thick frequency comb. We will say more about this in Sect. 5.4.

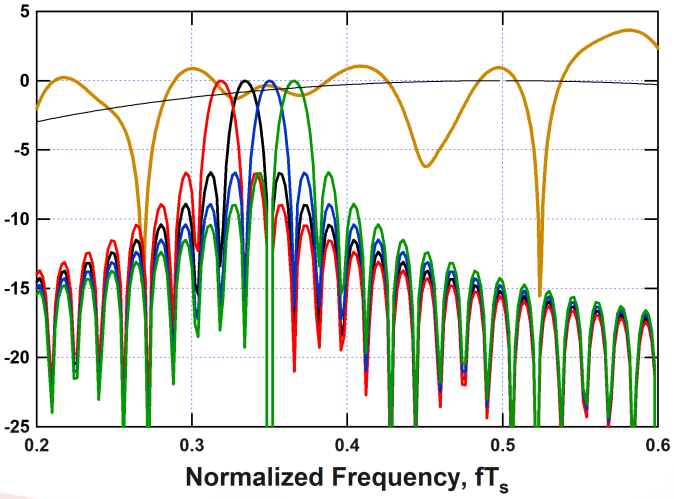


Figure 5.3 The principle of Multicarrier Signaling

5.2 From Generic Multicarrier to OFDM

5.2.1 The Multicarrier Modulator

In the previous, heuristic discussion some important aspects were not specified, such as the frequency-spacing between subcarriers, and the way to efficiently build such a complicated signal format.

To go on with the description of the multicarrier format, consider Fig. 5.4, in which the input symbols belong to the alphabet of an arbitrary linear modulation (typically M-PSK or M-QAM) with an input symbol rate R_s. The (high-rate) stream of these symbols is *parallelized* (demultiplexed) into N sub-channels with a common, "slower" multicarrier symbol rate $R_M = R_s/N$. This parallelization intrinsically creates a segmentation of the input data into blocks of N input symbols, for a duration of the block equal to the so-called *multicarrier symbol time* T_M. The expression of the multicarrier digital signal is best understood if we adopt a two-index notation: with $k = 0, 1, ..., N-1$ we indicate the number (identifier) of a generic subcarrier at frequency $f_k = k \cdot f_{sc}$, where fsc is the subcarrier spacing. Also, with m we denote the index of the m-th multicarrier symbol block, i.e., extending from time mT_M to $(m+1)T_M$. Consider now the scheme of the multicarrier modulator shown in Fig. 5.4. As is seen, the signal is the sum of the N components $y_k(t)$, where each component is obtained by modulating a baseband digital signal $x_k(t)$ on the the k-th subcarrier with frequency $f_k \triangleq k \cdot f_{sc}$:

$$x_{MC}(t) = \sum_{k=0}^{N-1} y_k(t) = \sum_{k=0}^{N-1} x_k(t)\, e^{j2\pi k f_{sc} t} \tag{5.6}$$

In its turn, the k-th baseband digital signal $x_k(t)$ is

$$x_k(t) = \sum_{m=-\infty}^{+\infty} c_m^{(k)} p(t - mT_M) \tag{5.7}$$

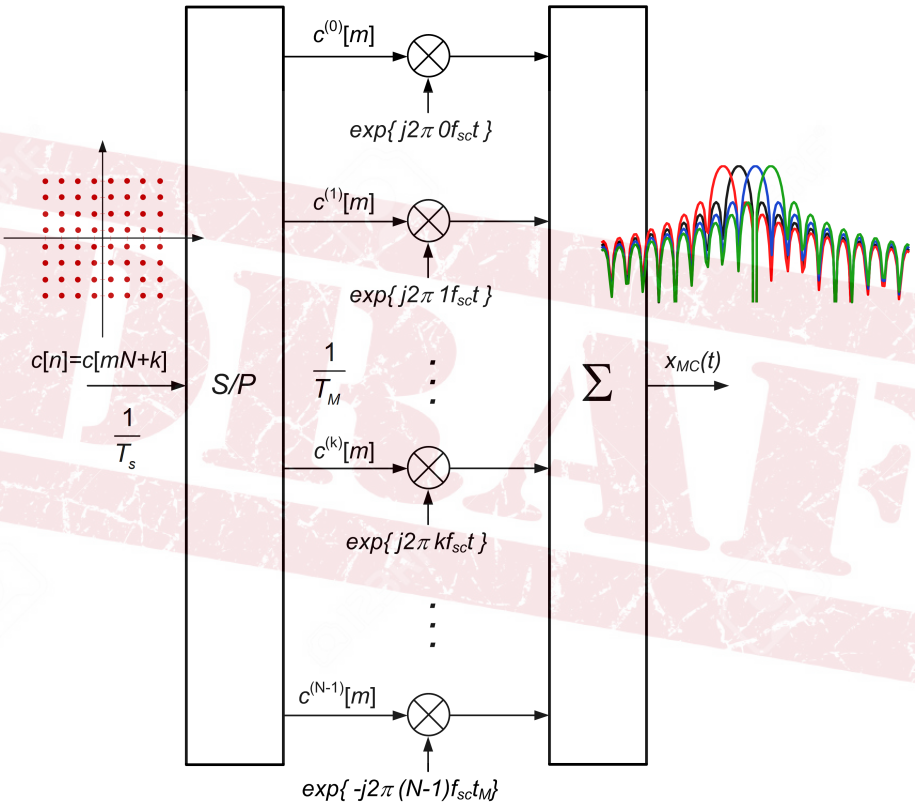


Figure 5.4 Schematic of a Multicarrier Modulator

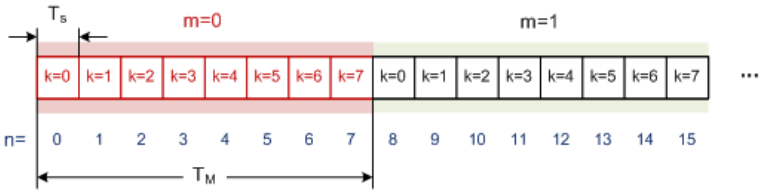


Figure 5.5 Relation between different symbol indices n, k, m with $N = 8$

where $c_m^{(k)}$ is the constellation symbol that is mapped onto the subcarrier number k at time mT_M, and where $p(t)$ is the NRZ (rectangular) multicarrier symbol pulse "marking" the symbol time, and extending with unit amplitude between $t = 0$ and $t = T_M$. The relation between the input symbols c_n running at the rate R_s and the symbol $c_m^{(k)}$ running at the rate $R_M = R_s/N$ on the $k-$th subcarrier is simple: considering that $0 \leq k \leq N-1$, we have

$$c_m^{(k)} = c_{mN+k}$$

The relation between the input and the multicarrier data symbols is graphically represented in Fig. 5.5 for $N = 8$ and clearly shows the difference in the clock rate between the two.

All in all, the complete expression of the multicarrier signal $x_{MC}(t)$ is

$$x_{MC}(t) = \sum_{k=0}^{N-1} \sum_{m=-\infty}^{+\infty} c_m^{(k)} p(t - mT_M)\, e^{j2\pi k f_{sc} t} \tag{5.8}$$

5.2.2 Multicarrier Demodulator and OFDM

The idea leading to the particular format (5.8) of the multicarrier signal is very clear. Of course, we have also to show that the data symbols $c_n = c_{mN+k} = c_m^{(k)}$ can be actually recovered for the signal once it comes to the receiver. In particular, putting different symbols on different subcarriers with no particular specification on the value of the subcarrier spacing may intuitively create an undesired phenomenon of *inter-carrier interference* (ICI) between adjacent carriers, if they are too close to each other to be separated – this ICI issue has to be closely looked into.

Assume that subcarrier # k is the only one that is transmitted on the wireless channel, and no other subcarrier is actually active at the same time. In this case, the structure of the optimum receiver on the AWGN channel is that shown in Fig. 5.6 – a correlation receiver with the bandpass waveform on frequency kf_{sc} or, if you wish, a demodulator to convert the received signal from (sub-)frequency kf_{sc}, followed by a detection filter matched to $p(t)$ in the form of an integrator over a multicarrier symbol time T_M. We have also made the assumption that we intend to estimate the value of symbol at multicarrier time 0, i.e., $m = 0$, but generalization to any m is trivial and leads to the same structure and results.The output (soft decision variable for symbol $c_0^{(k)}$ is

$$z_0^{(k)} = \frac{1}{T_M} \int_0^{T_M} y_k(t) e^{-j2\pi k f_{sc} t} dt \tag{5.9}$$

The reader may wish to insert into (5.9) the expression of $y_k(t)$ coming form (5.8) and check that in the absence of noise $z_0^{(k)} = c_0^{(k)}$.

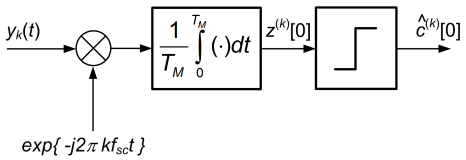


Figure 5.6 Optimum receiver for the k-th subcarrier

What if now we activate another frequency in the raster of the N subcarriers in the signal, say, # i ? The input signal to the receiver in Fig. 5.6 will have another term in addition to the same $z_0^{(k)}$ as above, let us call it $I_0^{(i,k)}$ meaning that it represents (at time $m = 0$) the interference generated by subcarrier i at the output of the receiver for subcarrier k. Let us compute this term considering that, for $0 \le t \le T_M$, $y_i(t) = c_0^{(i)} e^{j2\pi i f_{sc} t}$:

$$I_0^{(i,k)} = \frac{1}{T_M}\int_0^{T_M} c_0^{(i)} e^{j2\pi i f_{sc} t} \cdot e^{-j2\pi k f_{sc} t} dt = c_0^{(i)} \frac{e^{j2\pi(i-k)f_{sc}T_M} - 1}{j2\pi(i-k)f_{sc}T_M} \tag{5.10}$$

We see that in general this term is different from 0, and is sizeable when f_{sc} is small with respect to $1/T_M$ and i is close to k – the subcarriers are "too close" to each other.

A nice case would be that for which $I_0^{(i,k)} = 0$ irrespective of the values of i and k. This happens for instance when $f_{sc} >> 1/T_M$ or when $f_{sc} = q/T_M$, q any integer. This condition of no ICI in the receiver is highly desirable, but what value of q should we select? A large value of q means a large subcarrier spacing; as we will see in detail later on, large spacing means that our signal spans (occupies) a large bandwidth with its raster of subcarriers, thus diminishing the spectral efficiency of the link. The "right" value of q is therefore the *minimum* that still guarantees no ICI, or, in mathematical words, *orthogonality* between carriers: q=1, and $I_0^{(i,k)} = 0$ for any $i \neq k$.

When q=1, $f_{sc} = 1/T_M$, i.e., the subcarrier spacing is exactly equal to the signaling rate R_M on the subcarriers, and the multicarrier signal is called OFDM (Orthogonal Frequency-Division Multiplexing):

$$x_{OFDM}(t) = \sum_{k=0}^{N-1} \sum_{m=-\infty}^{+\infty} c_m^{(k)} p(t - mT_M)\, e^{j2\pi kt/T_M} \tag{5.11}$$

Thanks to the orthogonality condition, the optimum OFDM receiver for the AWGN is just a stack of N optimum receivers like the one in Fig. 5.6, each one tuned on its own subcarrier $f_k = k/T_M$, $k = 0, 1, ..., N-1$, and globally not affected by any ICI.

5.2.3 Spectral Efficiency of OFDM

Once the format of the signal was established according to the favorable orthogonality condition, we have to check that such format is spectrally efficient in terms of required (bit/s)/Hz – in other terms we have to find the bandwidth occupancy of $x_{OFDM}(t)$.

To do this, we take back into consideration the expression introduced in (5.6), that also applies to OFDM:

$$x_{MC}(t) = \sum_{k=0}^{N-1} y_k(t) = \sum_{k=0}^{N-1} x_k(t)\, e^{j2\pi kt/T_M} \tag{5.12}$$

and where $x_k(t)$ is still as in (5.7). From the latter equation we can easily see that $x_k(t)$ is statistically independent of $x_i(t)$, $i \neq k$ since the data symbols in the two signals are different; this property is also inherited by $y_k(t)$ and $y_i(t)$, facilitating the computation of the power spectral density (psd) $S_{OFDM}(f)$ of $x_{OFDM}(t)$. Thanks to the independence in fact

$$S_{OFDM}(f) = \sum_{k=0}^{N-1} S_{y_k}(f) \tag{5.13}$$

and, going on with the computation,

$$S_{OFDM}(f) = \sum_{k=0}^{N-1} S_{y_k}(f) = \sum_{k=0}^{N-1} S_{x_k}\left(f - \frac{k}{T_M}\right) \tag{5.14}$$

Consider now that the psd of the baseband data signal $x_k(t)$ is

$$S_{x_k}(f) = \frac{C_2}{T_M} \cdot |P(f)|^2 = C_2\, T_M \mathrm{sinc}^2(fT_M)$$

where $C_2 \triangleq E\left\{|c_n|^2\right\}$, $P(f)$ is the Fourier transform of $p(t)$, and where no dependence form k is actually left. The conclusion is that

$$S_{OFDM}(f) = C_2\, T_M \sum_{k=0}^{N-1} \mathrm{sinc}^2\left(\left(f - \frac{k}{T_M}\right) T_M\right) \tag{5.15}$$

confirming the intuition that the total spectrum of OFDM, as anticipated in Fig. 5.3 is the superposition of N equal elementary spectra, each belonging to the stream on a certain subcarrier, and of course each centered on the relevant subcarrier frequency.

The appearance of the OFDM spectrum is shown in Fig. 5.7. As is seen, the spectrum is not bandlimited, and rolls-off with frequency faster for a higher number of subcarriers. The reference in terms of spectral efficiency is the good-old Square-Root Frequency Raised-Cosine (SRFRC) pulse shape for linear modulation, whose modulation RF bandwidth for a signaling interval T_s is equal to $B_{SRFRC} = (1+\beta)/T_s$ ($0 \leq \beta \leq 1$ is the roll-off factor. What is now the RF bandwidth for our OFDM spectrum ? The actual OFDM spectrum, resulting form the superposition of infinite-bandwidth sinc spectra is infinite-bandwidth – we have to adopt a practical definition of spectrum occupancy, like -3 dB bandwidth or similar. [1] Sticking to this definition, we can see from Fig. 5.7 that the modulation RF bandwidth is $B_{OFDM} \simeq 1/T_M$, just like using the SRFRC spectrum with $\beta = 0$, the highest spectral frequency that we can obtain: the Nyquist bandwidth – another positive feature of OFDM.

5.3 DSP-based OFDM Modem

5.3.1 Digital OFDM Modulator

OFDM has become a ubiquitous technology when it comes to wideband wireless digital communications. IEEE 802.11 Wi-Fi for wirelss Local Area Network, European Digital

[1] The conclusion that we draw here is not changing so much is we adopt a more restrictive definition, like -10 or -20 dB bandwidth – there will just be a small diminution in the spectral efficiency, just like having a very small value of the roll-off value in the SRFRC spectrum

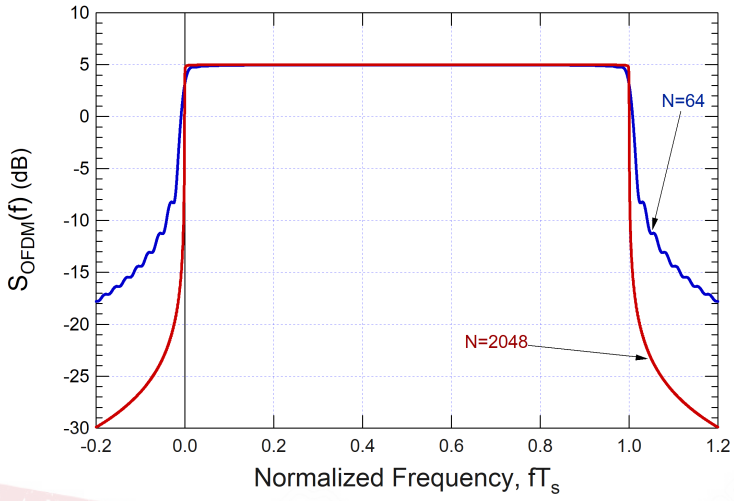


Figure 5.7 Power Spectral Density of the OFDM Signal

Terrestrial Television DVB-T, 4th Generation (4G) cellular networks LTE, 5G's NR (New Radio) interface, are examples of the adoption of OFDM technologies, with a number of carriers ranging from 64 (Wi-Fi) to 32k (DVB-T2). The complexity of building an OFDM modem with 32k modulators in the transmitter and/or 32k demodulator/filters in the demodulator appears formidable, but (baseband) Digital Signal Processing comes to the aid in this respect.

Examine the structure of the OFDM signal in the interval $0 \leq t < T_M$, i.e., in the first OFDM symbol: the general expression 5.11 can be simplified to (remember that $p(t) \equiv 1$ in that time interval)

$$x_{OFDM}(t) = \sum_{k=0}^{N-1} c_0^{(k)} p(t)\, e^{j2\pi kt/T_M} = \sum_{k=0}^{N-1} c_0^{(k)}\, e^{j2\pi kt/T_M} \tag{5.16}$$

Assume now that in our DSP-based modem we intend to generate *samples* of this signal, taken with a certain sampling frequency $f_{sa} = 1/T_{sa}$ to be specified, as follows:

$$x[n] \stackrel{\triangle}{=} x_{OFDM}(nT_{sa}) = \sum_{k=0}^{N-1} c_0^{(k)}\, e^{j2\pi knT_{sa}/T_M} \tag{5.17}$$

What is a good choice of f_{sa} ? For a baseband signal with a spectrum limited into the bandwidth B, we know that $f_{sa} \geq 2B$. In our case, we may not be so confident about the value of B to be considered. The doubt comes form the fact that the spectrum of our OFDM signal shown in Fig. 5.7 is not symmetric with respect to the 0 frequency – we can say that it is shifted around a sort of intrinsic carrier frequency of value $1/(2T_s)$. So true, that when we wish to upconvert our baseband signal to a radio channel whose pass band is symmetric and centered around a carrier frequency f_{RF}, the I/Q modulator *must* use a local oscillator at frequency $f_0 - 1/(2T_s)$. If we wish to apply to our case the usual Nyquist's rule, we have to consider a symmetric version of the OFDM spectrum, like in Fig. 5.8 The conclusion is that the right value for B to be used in Nyquist's rule is $B_{sym} = 1/(2T_s)$ so that, taking the

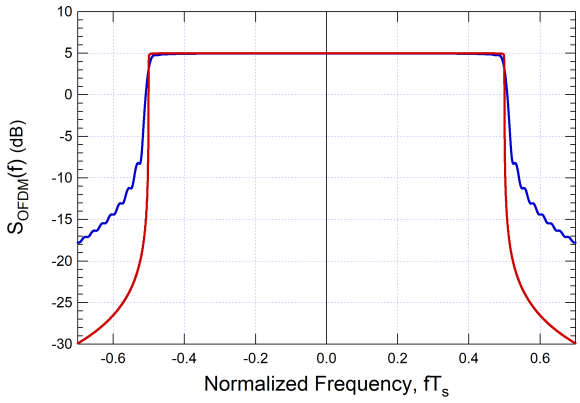


Figure 5.8 Symmetric psd of OFDM

minimum value, we can set $f_{sa} = 2B_{sym} = 1/T_s$. Using this value into 5.17, and recalling that $T_M = NT_s$ we get

$$x[n] \triangleq x_{OFDM}(nT_{sa}) = \sum_{k=0}^{N-1} c_0^{(k)}\, e^{j2\pi knT_s/T_M} = \sum_{k=0}^{N-1} c_0^{(k)}\, e^{j2\pi nk/N} \quad n = 0, 1, ..., N-1 \tag{5.18}$$

This tells us that the algorithm that a DSP processor has to compute on the (complex-valued) input symbols $c_0^{(k)}$ to get the (complex-valued, I/Q) samples of the OFDM signal $x[n]$ is just an Inverse Discrete Fourier Transform (IDFT), that can be efficiently implemented in SW/HW via well-know *Fast* algorithms/architectures (FFT) – another positive feature of OFDM.

5.3.2 Digital OFDM Demodulator

Can we do the same for the demodulator in Fig. 5.6? We assume that the input signal of the demodulator, instead of being the analog I/Q baseband received signal $r(t)$ is the result of ADC conversion of the latter, i.e., $r[n] \triangleq r(nT_s)$ where we have kept the same sampling frequency as above $f_{sa} = 1/T_s$. In 5.6 the decision variable $z_0^{(k)}$ is derived as in (5.9); assuming that $r(t)$ is strictly bandlimited, the result of the integral can be exactly computed by the *sum* of the signal samples on the same time interval, scaled by the sampling interval T_s that takes the place of the differential dt, so that we have

$$z_0^{(k)} = \frac{1}{T_M} \sum_{n=0}^{N-1} r[n]\, e^{-j2\pi knT_s/T_M} T_s = \sum_{n=0}^{N-1} r[n]\, e^{-j2\pi nk/N} \tag{5.19}$$

or, an algorithm of (direct) DFT, as may be expected: in the modulator, the incoming symbols, one for each frequency, are turned into OFDM time-samples through an *inverse* Fourier transformation; in the demodulator, the incoming time samples of the received signal are converted to the frequency domain by a *direct* Fourier transformation, to find again the (frequency-domain) constellation symbols. The same consideration about efficiency of HW/SW implementation that we made for the modulator also apply to the demodulator – the core of the OFDM modem is an FFT processor.

5.3.3 Virtual Carriers

There is an aspect in the development of the DSP-based OFDM modem that we have a bit overlooked. We assume to process digitally a signal that is not bandlimited. To understand the real implication of this, let us try to derive the psd $\bar{S}_{OFDM}(f)$ of the sampled OFDM signal with $f_{sa} = 1/T_s$ as in the subsection above. As is known, when sampling we introduce replicas in the frequency domain, so that

$$\bar{S}_{OFDM}(f) = \sum_{k=0}^{N-1} S_{OFDM}\left(f - \frac{k}{T_s}\right)$$

The situation is represented in Fig. 5.9 (a) where the original spectrum of the analog signal is copy-pasted around multiples of the sampling frequency $1/T_s$. It is apparent that some amount of aliasing is introduced by the operation of sampling just because of the "tails" of the spectrum beyond 0 and $1/T_s$. Reducing the aliasing can be done by increasing the sampling frequency (thus spacing more apart the spectrum replicas). This is not convenient since changing the sampling frequency means altering the algorithms in (5.18) and (5.19), in particular losing the nice property of representing (I)DFT/FFTs. The alternative is reducing the bandwidth of the signal – the right choice. Spectral occupancy reduction can be easily obtained by a trick: leaving the modulation algorithm unchanged (5.18), what is to be done is delivering to the modulator a block of symbols containing a trailing run of zeros, i.e., segmenting the incoming data symbols into block of $N - N_v$ values instead of N, and assuming that the last N_v symbols are zero. The modulator receives this zero-padded vector (block) of symbols and computes the order-N DFT as before – the result is that the last N_v subcarriers are *virtual*, and so the power spectrum has a narrower bandwidth, extending only up to the frequency $(N - N_v)/T_M = N_a/N \cdot 1/T_s$, where $N_a = N - N_v$ is the number of *active* subcarriers, as shown in Fig. 5.9 (b):

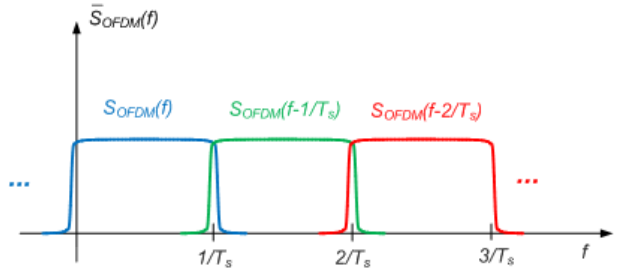
$$\bar{S}_{OFDM}(f) = \sum_{k=0}^{N-N_v-1} S_{OFDM}\left(f - \frac{k}{T_s}\right)$$

When sampling at the same frequency $1/T_s$, the situation is now that in Fig. 5.9 (c) with no aliasing.

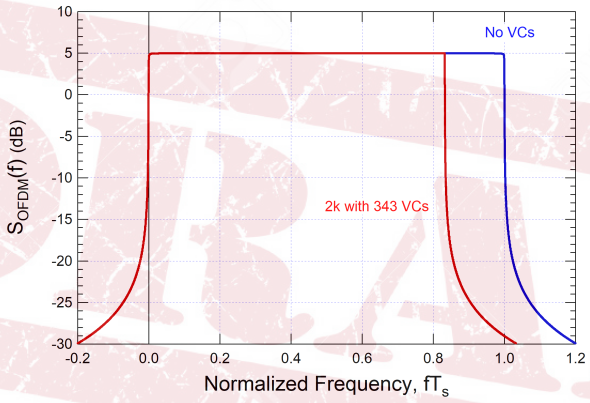
This trick of the virtual carriers is always used in OFDM formats. Just to make an example, the two basic transmission modes of digital European terrestrial television DVB-T are i) the 2k mode with N=2048 (2k mode), T_M=224 μs, and ii) the 8k mode with N=8192, T_M=896 μs, and N_v=1375. The reduction of bandwidth occupancy is similar in the two modes and amounts to $(N - N_v)/N$=0.83. Contrary to the first appearance, there is no loss in using virtual carriers: it is true that $N - N_v$ potentially available symbols (subcarriers) are not exploited and the transmitted bit rate is reduced wrt the maximum potentially available, but the occupied bandwidth of the signal is reduced correspondingly, so that the spectral efficiency stays the same.

5.4 Frequency-Domain Equalization of OFDM

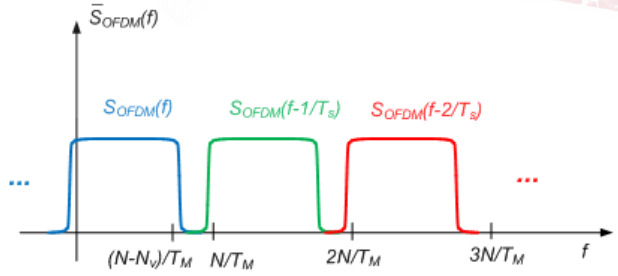
The OFDM signal format was invented to create a wideband, high bit-rate signal with intrinsic robustness against strong frequency selectivity. Can we confirm that the intuition leads to correct results when actually sending out (5.11) into a multipath channel? Shall



(a)



(b)



(c)

Figure 5.9 Aliasing and Virtual Carriers in OFDM

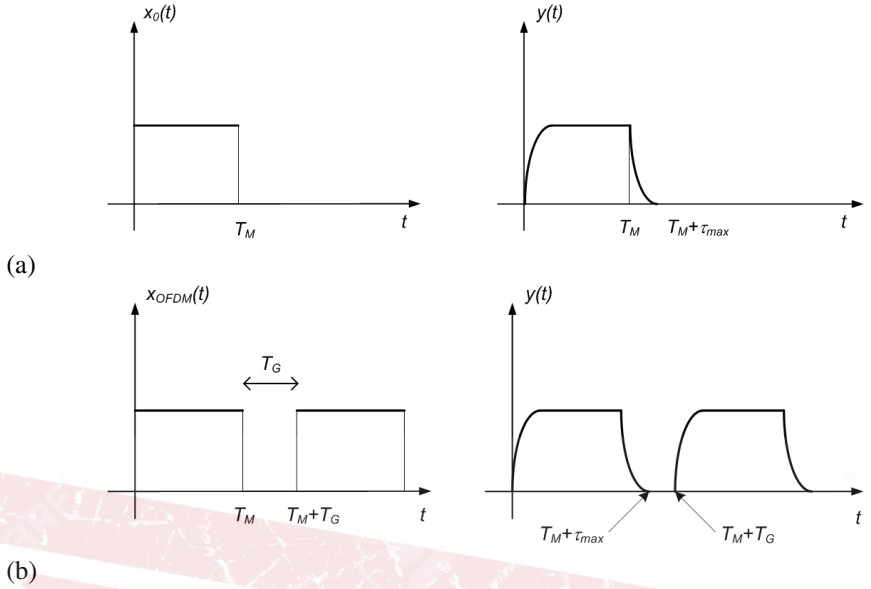


Figure 5.10 IBI and Guard Time in OFDM

we change somewhat in the architecture of the FFT-based receiver that we devised from the optimum single-carrier receiver on the AWGN channel (with no mention to channel distortions...)?

5.4.1 Cyclic Prefix

When we send out an OFDM signal onto a multipath channels, it undergoes modification via the channel impulse response $h(t)$ in the well-know multiple-echo form (4.4). If we assume that we are transmitting a waveform $x_0(t)$ containing just *one* OFDM symbol (5.16), that is confined into the OFDM symbol time interval $0 \leq t < T_M$, we understand that the effect of the multipath will be that of *smearing* the stricly time-limited waveform representing this symbol into something that will extend up to $T_M + \tau_{max}$ where τ_{max}, usually equal to τ_{N_r} in 4.4) is the maximum echo delay (i.e., the maximum differential delay in the multipath model) – this is sketched in Fig. 5.10 (a). The lengthening of the OFDM symbol because of multipath has the effect to create what is called Inter-Block Interference (IBI), or in-other words, inter-OFDM-symbol interference. With the orthogonality condition we have enforced that no inter-symbol interference (ISI) is experienced in the receiver across the symbols composing one block (one OFDM symbol). But now, with the effect of multipath, self-interference pops out again in the form of IBI, that can be equally detrimental.

A solution to this issue is shown in Fig. 5.10 (b) (it is kind of Columbus' egg): to prevent IBI, what we do is sending out the subsequent OFDM symbol (m=1) not immediately after the previous one is over (not at time T_M), but after a further time period T_G has elapsed (i.e., at time $T_M + T_G$), so that the "tail" of the previous block caused by multipath has faded out. This additional time is called *guard time* and prevents IBI altogether – the only constraint is that $T_G \geq \tau_{max}$.

It is clear that the use of the guard interval decreases the spectral efficiency of the system: since it is necessary to maintain a constant information rate, introducing a guard interval different from 0 means being forced to reduce the net OFDM symbol time, and therefore increasing bandwidth occupancy for the same bit-rate – the longer is the guard time, the less efficient will be the signal. We will quantify the loss later on – suffice it to say now that in practical OFDM formats the duration of the OFDM symbol is relatively large because of the high number of subcarrier that is used. Take for example the 2k mode of DVB-T: we have T_M= 224 μs. On the other hand, we know that the maximum delays that occur for a multipath channel are of the order of a few tens of μs (see Tab. 4.1) – the loss of efficiency is therefore modest. The trick of the guard time cannot be used in conventional single-carrier modulations because in that case the duration of the (conventional) constellation symbol T_s is fully comparable with that of the channel response, and the loss would be intolerable. [2]

In addition to efficiency loss, the guard time approach has a further disadvantage: remember that our FFT-based OFDM demodulator processes a vector of N I/Q signal samples collected in an OFDM symbol time T_M. When we have multipath, the length of the received signal, including the tail caused by multipath, is longer than T_M, so that in general the receiver should process $N + N_G$ samples, where $NG = T_G/T_s$ is the length of the guard time expressed in signal samples unit – this cannot be done without altering the FFT size and further complicating signal processing. By slightly modifying the format of the transmitted signal, however, it is possible to obtain optimum demodulation of the received signal via FFT processing of N samples with no IBI in spite of the multipath channel.

The idea is the following: sending out an isolated OFDM symbol into the channel has the effect of lengthening the received signal. On the contrary, assume (just for the moment) that the transmitted signal is not just the isolated symbol with m=0, but on the contrary is the *repetition* $x_p(t)$ of such symbol, so as to make the signal *periodic* of period T_M: $x_p(t) = \sum_m x_0(t - mT_M) = x_p(t + T_M)$. In such a situation, the output of the multipath channel $y_p(t)$ will be periodic as well, with the same repetition period like in Fig. 5.11 (a), and the receiver will only have to observe one period (a time span T_M) to perform optimum reception – one period corresponds again to N samples, and the hope is to be able to do FFT processing again.

Before discussing how this is related to real-world transmission of the actual OFDM signal (not periodic by any means), let us analyze our hypothesized situation further. Since the transmitted signal $x_p(t)$ is periodic, it can be expanded into a Fourier series:

$$x_p(t) = \sum_k X_k \, e^{j2\pi kt/T_M} \tag{5.20}$$

where X_k is the Fourier coefficient of the signal,

$$X_k \triangleq \frac{1}{T_M} \int_0^{T_M} x_p(t) \, e^{-j2\pi kt/T_M} \, dt \tag{5.21}$$

[2]This feature of being able to easily handle channels with large "echo" delays allows to solve an important issue in a broadcasting network. Robustness to the "natural" echoes of multipath based on the guard time works also in the case in which the echo comes from a *different repeater* of a broadcasting network operating on the same frequency (we call it "artificial multipath"). This greatly simplifies the design and operation of a so-called single frequency networks (SFN) for the distribution of a radio or television channel on a single frequency regional or national broadcast. The only difference is that the "artificial" multipath delays of an SFN are generally much greater than the "natural" delays of the multipath, and therefore the guard interval requirement must be adapted to the need.

By comparing (5.20) with the expression (5.16) of $x_0(t)$, we see that X_k does not actually need to be computed: by inspection, $X_k = c_0^{(k)}$, $k = 0, 1, ..., N-1$ (and 0 outside this interval). When $x_p(t)$ is sent into the wireless multipath channel with impulse response $h(t)$, the periodic output is $y_p(t) = x_p(t) \otimes h(t)$. The Fourier coefficient of $y_p(t)$ is $Y_k = X_k \cdot H(k/T_M)$ where $H(f)$ is the frequency repones of the channel, and k/T_M is the k-th harmonic frequency of the periodic signal, that by the way coincides with the k-th subcarrier frequency.

Good to know, but what is the relation of Y_k to our problem? Let us take back into consideration the OFDM demodulator 5.6 whose decision variable is

$$z_0^{(k)} = \frac{1}{T_M} \int_0^{T_M} y_p(t) e^{-j2\pi kt/T_M} dt \tag{5.22}$$

We see that the computation performed by the optimum demodulator is just... the computation of Y_k. The conclusion is that by implementing an optimum receiver on the periodic signal and on one OFDM symbol time T_M in the form of an FFT processor on N signal samples, what we get is

$$z_0^{(k)} = Y_k = X_k \cdot H(k/T_M) = c_0^{(k)} \cdot H\left(\frac{k}{T_M}\right) \quad k = 0, 1, ..., N-1 \tag{5.23}$$

The decision variable contains the k-th transmitted data symbol *with no ICI* as expected (orthogonality is preserved), and we can also clearly see the effect of the frequency-selective channel: th k-th data symbol is just amplitude/phase modified by the value of the frequency response of the channel at the very subcarrier frequency on which the symbol has been modulated – a full confirmation to our intuition, and the final proof that the invention of the multicarrier format actually works. We just lack the final detail: how is our nice result (5.23) related to the real-world case wherein the OFDM signal is *not* periodic? The answer is the following: we have to make the OFDM signal *look* periodic as much as possible, and the way to do this is to fill the guard time at the transmitter not just with... nothing, but with a suited periodic extension of the OFDM symbol $x_0(t)$. This is shown in Fig. 5.11 (b) where the guard time is filled with a *cyclic prefix*: the final section of the OFDM symbol waveform, i.e., the final N_G samples, are shifted and placed in front of the N samples of the waveform in the original OFDM symbol time to create the prefix. This is something that is done quite easily in the digital modulator – you just need to buffer the N samples of the waveform $x_0(t)$ of one symbol, pre-pend the cyclic prefix, and then send the $N + N_G$ samples to the DAC.

The cyclic prefix creates a pseudo-periodicity in the transmitted signal as well as in the received signal that is shown in Fig. 5.11 (b) where it is compared with the theoretical actual periodicity of $y_p(t)$. For a small time section the two signals (i.e., the actual OFDM signal with cyclic prefix and the theoretical periodic signal $y_p(t)$) look very much the same, so that the result of the FFT processor in the receive will just be the one computed on $y_p(t)$, namely, (5.23) as we wish.

While virtual carriers do not change the spectral efficiency of the signal, the cyclic prefix does. To be more specific, we have to revise the relation between the input symbol rate $1/T_s$ and the other quantities (OFDM symbol time, subcarrier spacing, bandwidth) of the OFDM signal. With virtual carriers, the input block time, instead of NT_s becomes $(N - N_v)T_s = N_a T_s$. To keep the pace and conserve real time, this must also be the time of a whole OFDM symbol *plus* the guard time (cyclic prefix) T_G, so that $N_a T_s = T'_M + T_G$

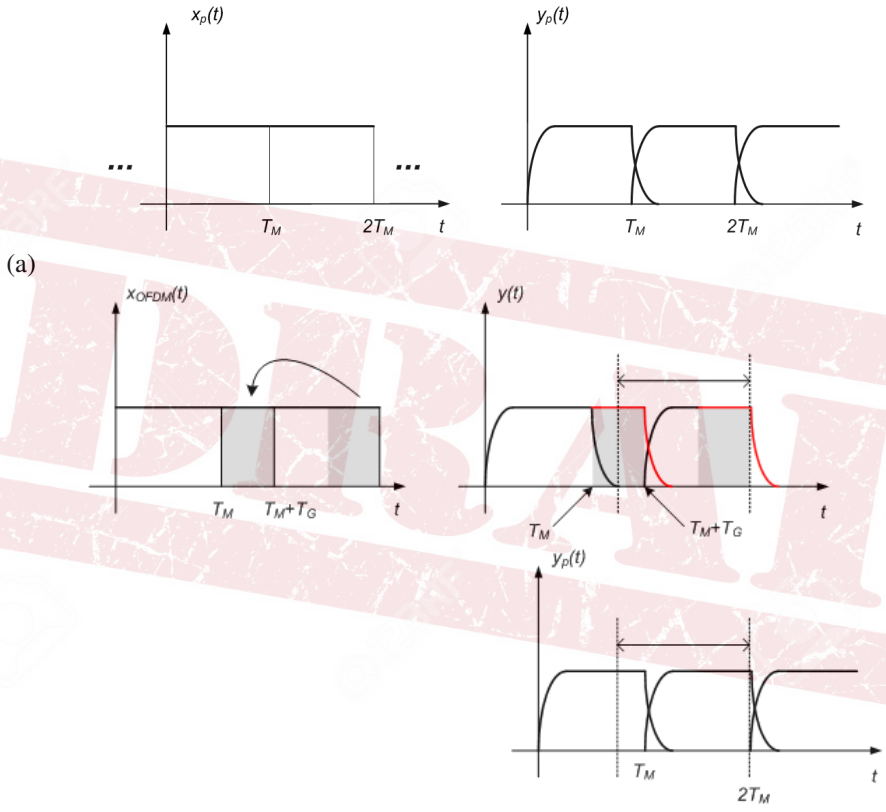


Figure 5.11 Cyclic Prefix in OFDM

where T'_M is the modified (shortened) duration of the OFDM symbol time. Since the new OFDM symbol time T'_M is shorter than before, the subcarriers, to keep orthogonality, will be spaced by the larger spacing $1/T'_M$, so that the new occupied bandwidth will be

$$B_{OFDM} = \frac{N_a}{T'_M} = \frac{N_a}{N_a T_s - T_G} = \frac{1}{T_s}\frac{N_a}{N_a - T_G/T_s}$$

We are tempted to say that $T_G/T_s = N_G$, but this is not true. In fact, the OFDM symbol length is shorter (now T'_M), so the sampling frequency in the modem has to be faster than before, and precisely $T_{sa} = N/T'_M$ (N time samples in one OFDM symbol). Therefore,

$$N_G = T_G/T_{sa} = NT_G/T'_M = \frac{NT_G}{N_a T_s - T_G} \quad \Rightarrow \quad T_G = N_G \frac{N_a T_s}{N + N_G}$$

Putting everything together,we finally get

$$B_{OFDM} = \frac{1}{T_s}\frac{N_a}{N_a - N_G \frac{N_a}{N+N_G}} = \frac{1}{T_s}\left(1 + \frac{N_G}{N}\right) \tag{5.24}$$

showing that the cyclic prefix increases the bandwidth, for the same input symbol rate, by a factor $1 + N_G/N$ as expected. A more complicated computation is needed to derive the explicit expression of the psd of the OFDM signal with cyclic prefix. The result is

$$S_{OFDM}(f) = C_2 \left(T_G + T'_M\right) \sum_{k=0}^{N-N_v-1} \text{sinc}^2\left(\left(f - \frac{k}{T'_M}\right)\left(T_G + T'_M\right)\right) \tag{5.25}$$

and the explanation of the formula is clear: the subcarrier spacing (and so the spacing between the sinc functions) has changed to $1/T'_M$, widening the total spectrum occupancy as in (5.24), but the bandwidth of each subcarrier is the same as before, since $(T_G + T'_M)$ is equal to old T_M. The result is an almost unnoticeable "ripple" in the passband of the spectrum.

Example 5.39

Just to make an example of all of the parameters that we have described, take the IEEE standard called 802.11g, one of the possible variants of the well known technology commercially called Wi-Fi. It is based on N=64 OFDM, with 48 data carriers, 4 pilot carriers, N_v=12 virtual carriers, and N_G=16 cyclic prefix length. The OFDM symbol time is $T'_M + T_G$=4 μs with T_G=0.8 ns and T'_M=3.2 μs, so that $f_{sc} = 1/T'_M$=312 kHz. The nominal occupied bandwidth is $52 * f_{sc}$=16.25 MHz. The possible constellations (same on all subcarriers) are B/QPSK, 16-QAM, 64-QAM, and the coding rate r is 1/2, 3/4 or (for 64-QAM only) 2/3.

Introduction of the cyclic prefix also mitigates another issue in the functioning of the receiver: time synchronization. The FFT processor must know in general the correct time epoch to start collecting the N samples corresponding to an OFDM symbol time – this time epoch is marked as t_e in Fig. 5.12. What happens if the receiver estimates that value with a certain error ε, so that it starts collecting samples at time $t_e - \varepsilon$ instead of the correct value t_e? Provided that $\varepsilon \leq T_G - \tau_{max}$ we can see that the waveform between $t_e - \varepsilon$

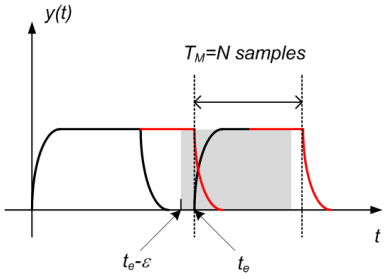


Figure 5.12 Time-Synchronization Error in OFDM

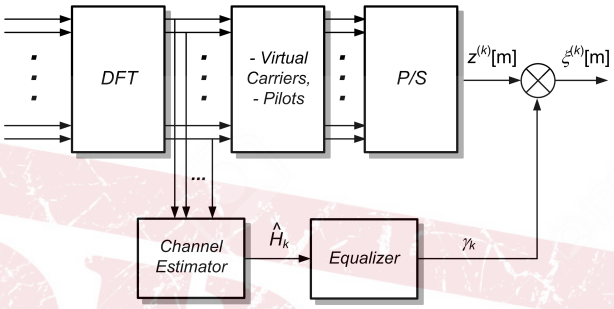


Figure 5.13 Compensation of Channel Frequency Response

and $t_e - \varepsilon + T_M$ is a *cyclic shift* of what is contained between ε and $\varepsilon + T_M$, i.e., it still represent one full period of the waveform $y_p(t)$, so that the receiver will still compute the "right" value of $Y_k = z_k^0$ – the only difference being a phase shift factor $\exp\{-j2\pi k\varepsilon/T_M\}$ caused by the delay ε. This factor can be considered as part of an "extended" frequency response on the channel, including this (unknown to the receiver) further phase shift, so that

$$z_0^{(k)} = c_0^{(k)} \cdot H_k \quad , \quad H_k \triangleq H(k/T_M)e^{-j2\pi k\varepsilon/T_M} \tag{5.26}$$

Of course, the effect of this factor H_k on data detection has to be taken into consideration. If the decision variable $z_0^{(k)}$ is sent as-is to an I/Q threshold with the standard decision regions for, say, M-QAM, such decision could be catastrophic because the constellation points are affected by an (unknown) scale factor $|H_k|$ and are rotated by an (unknown) phase shift $\angle H_k$. This is no different from what happens in a modem for conventional single-carrier modulation on a frequency-flat channel, the only difference being that such factors $|H_k|$ and $\angle H_k$, or if you wish the complex-valued channel coefficient H_k is *variable form subcarrier to subcarrier*, so that the receiver has to estimate N_a such factor across all subcarriers, and then use the estimated values to correct the N_a decisions $z_m^{(k)}$ at time m in order to adjust the constellation points prior decision, as suggested in Fig. 5.13. This is just the subject of the next subsection.

5.4.2 Channel Estimation and Compensation

Let us take back into consideration the ubiquitous AWGN that accompanies the received signal $y(t)$ out of the selective channel, so that our received signal will be $r(t) = y(t) + w(t)$

where as usual $w(t)$ is AWGN whose I/Q components are independent and both have psd $S_w(f) = N_0$. Considering the OFDM demodulator 5.6, it is easy to see that the decision variable $z_0^{(k)}$, in addition to the signal component already computed in (5.26), will be accompanied by a Gaussian noise component:

$$z_0^{(k)} = c_0^{(k)} \cdot H_k + w_0^{(k)} \tag{5.27}$$

where $w_0^{(k)}$ is a zero-mean complex-valued Gaussian random variable whose I/Q components are independent and both bear a variance $\sigma_w^2 = N_0/T_M$. So estimating the response of the selective channel H_k from the observation of $z_0^{(k)}$ is not feasible just because $c_0^{(k)}$ *and* $w_0^{(k)}$ are unknown to the receiver. Such estimation could be surely done in the unrealistic hypothesis in which the data symbols $c_0^{(k)}$ were known: in that case, we might assume that the noise would be negligible, and so the estimate would be

$$\hat{H}_k \simeq z_0^{(k)}/c_0^{(k)} \tag{5.28}$$

with an accuracy depending on the SNR in the receiver.

This is exactly a possible strategy to do *channel estimation* that is implemented into Wi-Fi: with a certain regularity, the transmitter, instead of sending out real data, transmits a known OFDM preamble, i.e., an OFDM waveform wherein all data symbols are known to the receiver. Assuming that the response of the channel is constat across a number of successive, consecutive OFDM symbols, we regard what derived after processing of the preamble a valid estimate until the next preamble is received.

Once the channel is known, what the receiver is supposed to do is to compensate for it. A naive but effective strategy is just removing the channel from the decision variable $z_0^{(k)}$ by simple (complex) division: $\xi_0^{(k)} = z_0^{(k)}/\hat{H}_k$. If we do so, we have in fact

$$\xi_0^{(k)} = \frac{z_0^{(k)}}{H_k} \simeq c_0^{(k)} + \frac{w_0^{(k)}}{H_k} = c_0^{(k)} + \nu_0^{(k)} \tag{5.29}$$

where we have assumed that the estimate is accurate, so that $\hat{H}_k \simeq H_k$. Equation (5.29) clearly shows that this criterion of channel compensation, called for historical reasons *Zero-Forcing (ZF) equalization*, actually works, in that the data symbol $c_0^{(k)}$ is restored with its own original amplitude and phase. From a different standpoint, we can say that ZF equalization, like the conventional equalizer $E(f)$ of section 5.1.1 tries to *invert* the frequency response of the channel to compensate for its distortion.

Equation (5.29) also shows that the data symbol is accompanied by a noise component $\nu_0^{(k)}$ whose variance is not the same as σ_w^2. In fact

$$\nu_0^{(k)} = \frac{w_0^{(k)}}{H_k} = \frac{w_0^{(k)} e^{-j\angle H_k}}{|H_k|} \quad \Rightarrow \quad \sigma_\nu^2 = \frac{\sigma_w^2}{|H_k|^2}$$

so that the variance of the noise component changes from subcarrier to subcarrier, and turns out to be large when $|H_k|$ is small: we may have a phenomenon of *noise enhancement* that we will consider later on when deriving the BER performance of the receiver. Transmitting the preamble to perform channel estimation of course introduces some overhead, that is, loss of efficiency of the link: is it the price to be paid to make the receiver work.

A different method to perform channel estimation without devoting a whole OFDM symbol to that function relies on the introduction of *pilot subcarriers*. A pilot subcarrier,

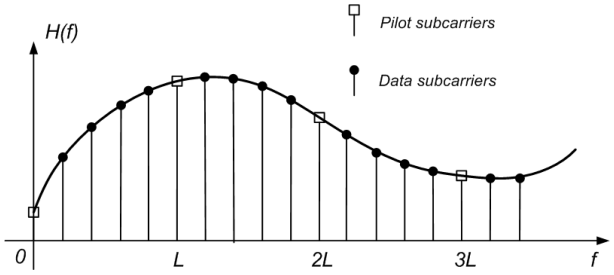


Figure 5.14 Interleaved Pilots for Channel Estimation

instead of carrying a valid data symbol from the transmitted constellation, contains a known value (the pilot symbol) dictated by the standard, for instance $(1 + \jmath 0)$. The pilot symbols are interleaved in frequency with the (valid) data subcarriers, for instance one pilot every 4 data carriers like in Fig. 5.14. The receiver performs channel estimation like in 5.28 in *every* OFDM symbol, but on the pilot subcarriers only. Fact is, this estimated value appears to be useless: we have no need to remove the channel apmlitude/phase from pilot subcarriers since we will never do data detection on those (the pilot symbols are known!). But, once we have H_K on all of the pilot subcarriers, we can use these values as *nodes* in an *interpolation formula* to reconstruct all of the missing H_k's on the data carriers. Often, piecewise polynomial or efficient spline interpolation is used – more refined approaches (like Wiener interpolation) require modeling of the variation of the channel response as a function of frequency and are outside the scope of this chapter

Example 5.40

Assume that the pilot carriers are regularly interleaved with data on a 1-in-L basis, i.e, after $L-1$ data carriers comes a pilot symbol. The overhead (loss) due to insertion of the pilots is $1/L$ and can be non-negligible – practical values for L are 4-5. The simplest non-trivial scheme to reconstruct the missing values of H_k is the *linear interpolator*. After finding two consecutive estimates, say $h_1 \triangleq \hat{H}_{\ell L}$ and $h_2 \triangleq \hat{H}_{(\ell+1)L}$, as in (5.28), the intermediate values $\hat{H}_{\ell L+1}, ..., \hat{H}_{\ell L+L-1}$ are found as suggested by Fig. 5.15:

$$\hat{H}_k = h_1 + \frac{h_2 - h_1}{L}(k - \ell L) \quad , \quad \ell L < k < \ell L + L \tag{5.30}$$

The accuracy of linear interpolation, as of any other algorithm, depends of course on the SNR in the receiver that affects the accuracy of the "initial" estimates $\hat{H}_{\ell L}$ and $\hat{H}_{(\ell+1)L}$on the nodes, and also by the *density* of pilots: the smaller is L the higher will be the accuracy of interpolation since the variation of the channel between two pilots will be limited. But, small L also entails a higher overhead, so when devising a standard format a good trade-off has to be found.

Once channel estimation is performed as above, ZF frequency-domain equalization as in (5.29) can be performed. We have already mentioned that the ZF criterion may create problems in the dynamics of $\xi_0^{(k)}$ when $\hat{H}_k$ is close to 0. This can be prevented by adopting different equalization criteria. In general, we can say that $\xi_0^{(k)} = z_0^{(k)} \cdot \gamma_k$ where γ_k depends on the criterion – if $\gamma_k = 1/\hat{H}_k$ we fall back to ZF equalization. With M-PSK modulations

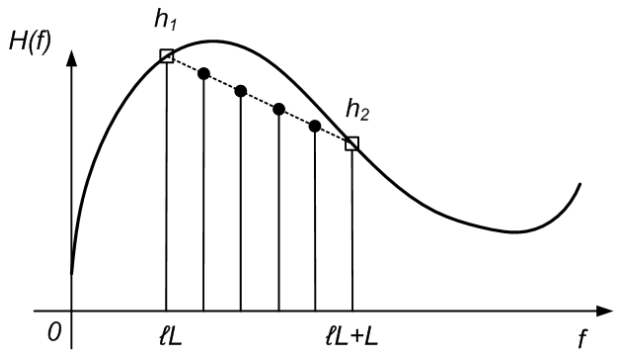


Figure 5.15 Linear Interpolation for Channels Estimation

we can also just compensate for the phase of the channel disregarding amplitude correction by doing $\gamma_k = \hat{H}_k^*$ – this kind of equalization looks very much like "matched filtering" to the channel response. Finally, we may wish to find γ_k such that the MMSE between $\xi_0^{(k)}$ and $c_0^{(k)}$ is minimum, also considering that $z_0^{(k)}$ contains the noise component $w_0^{(k)}$. If we do so, we end up with

$$\gamma_k = \frac{\hat{H}_k^*}{|\hat{H}_k|^2 + 1/(2E_s/N_0)}$$

where E_s/N_0 is the usual average signal-to-noise ratio at the receiver. When the SNR is very high, MMSE equalization falls back to ZF (in fact, $1/(2E_s/N_0) \to 0$ and $\gamma_k \to \hat{H}_k^*/|\hat{H}_k|^2 = 1/\hat{H}_k$); when on the contrary the SNR is very low, $\gamma_k \to \hat{H}_k^* \cdot (2E_s/N_0)$ and MMSE tends to matched filtering. In intermediate cases, it has anyway the effect of preventing the phenomenon of noise enhancement by regularizing the coefficient γ_k that never tends to infinity even if H_k is very small.

5.4.3 BER performance of OFDM with ZF Equalization

Let us take back into consideration expression (5.8) of the OFDM signal:

$$x_{OFDM}(t) = \sum_{k=0}^{N-1} y_k(t) = \sum_{k=0}^{N-1} x_k(t) e^{j2\pi kt/T_M} = \sum_{k=0}^{N-1} \sum_{m=-\infty}^{+\infty} c_m^{(k)} p(t - mT_M)\, e^{j2\pi kt/T_M} \tag{5.31}$$

where for simplicity we neglect virtual carriers and cyclic prefix. What is the average radio-frequency power of this signal? Since the $x_k(t)$, as we already know, are mutually independent, and since the factor $e^{j2\pi kt/T_M}$ has unit amplitude and does not change the subcarrier power,

$$P_{OFDM} = \sum_{k=0}^{N-1} P_{x_k} = N \cdot P_{x_k} = N \cdot P_x$$

where we have used the fact that the signal on any subcarrier k always bear a power $P_{x_k} \equiv P_x$. Considering the format (5.7) of the k-th subcarrier signal, and the NRZ shape of $p(t)$, we see that $P_x = C_2/2$ (where the factor 1/2 comes from scaling power from baseband to passband), so that finally $P_{OFDM} = NC_2/2$. If we wish to give the OFDM

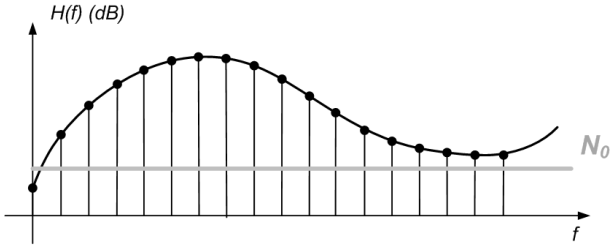


Figure 5.16 Variability of the SNR across subcarriers

signal a certain prescribed radio-frequency power level P_{RF}, then the correct expression of the OFDM radio signal will be

$$x_{OFDM}(t) = \sqrt{\frac{2P_{RF}}{NC_2}} \sum_{k=0}^{N-1} \sum_{m=-\infty}^{+\infty} c_m^{(k)} p(t - mT_M)\, e^{j2\pi kt/T_M} \tag{5.32}$$

This expression is expedient to correctly compute SNRs in any point of the TX/RX chain of the link. We can update equation (5.27) to include the new scaling factor as follows:

$$z_0^{(k)} = \sqrt{\frac{2P_{RF}}{NC_2}} c_0^{(k)} \cdot H_k + w_0^{(k)} = c_0^{(k)} \cdot H_k' + w_0^{(k)} \tag{5.33}$$

where H_k' is a new channel coefficient that includes the signal scaling factor. If we do ZF equalization, the result is again (with perfect channel estimation)

$$\xi_0^{(k)} = c_0^{(k)} + \nu_0^{(k)} \tag{5.34}$$

as in (5.29), but the variance of the noise components of $\nu_0^{(k)}$ is now

$$\sigma_\nu^2 = \sigma_w^2/|H_k|^2 \cdot \frac{NC_2}{2P_{RF}} = \frac{C_2 \cdot N_0}{2P_{RF}|H_k|^2 \cdot T_M/N} = \frac{C_2 \cdot N_0}{2P_r^{(k)} T_s} = \frac{C_2}{2E_s^{(k)}/N_0} \tag{5.35}$$

where $P_r^{(k)} = P_{RF}|H_k|^2$ is the received power on the k-the subcarrier, and it is different subcarrier by subcarrier, and $E_s^{(k)} = P_r^{(k)} T_s$ is the received energy per symbol on each subcarrier.

We see that each subcarrier operates in a different condition of SNR $E_s^{(k)}/N_0$, depending on the different channel amplitude $|H_k|$ – something that we already knew. Take for instance the representation of Fig. 5.16 where we sketched the frequency response of the channel and the noise psd level in dB – the "distance" between the channel response and the noise level is representative of the $E_s^{(k)}/N_0$ level on each subcarrier, that can be considerably variable.

Assessing the effect of channel variability on the modem BER is easy if we assume 4-QAM (QPSK) signaling, with symbols belongin to the alphabet $\{\pm 1 \pm \jmath\}$. In this case, $C_2 = 2$ and the BER is easily found by concentrating on either the I (or Q) component of $\xi_0^{(k)}$:

$$\xi_{0,I}^{(k)} = c_{0,I}^{(k)} + \nu_{0,I}^{(k)} = \pm 1 + \nu_{0,I}^{(k)} \tag{5.36}$$

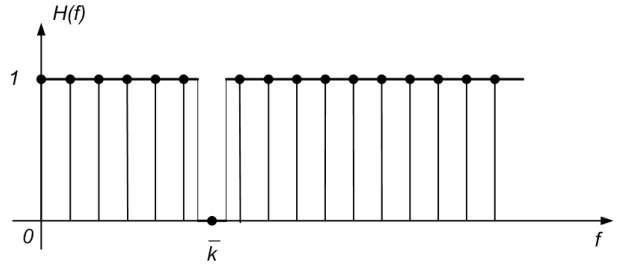


Figure 5.17 Quasi-everywhere good channel

where, from (5.35), the variance of the noise component is $\sigma_\nu^2 = (E_s^{(k)}/N_0)^{-1}$, so that the required BER is

$$BER_k = \mathrm{Q}\left(\sqrt{\frac{E_s^{(k)}}{N_0}}\right) = \mathrm{Q}\left(\sqrt{\frac{2E_b^{(k)}}{N_0}}\right) \tag{5.37}$$

We have highlighted the fact that the BER does depend on the subcarrier number k through the SNR $E_s^{(k)}/N_0$. The overall BER of the link will be the average of the different BER_k since the input information bits of the OFDM modem c_n distribute evenly across the various subcarriers:

$$BER = \frac{1}{N}\sum_{k=0}^{N-1} BER_k = \frac{1}{N}\sum_{k=0}^{N-1} \mathrm{Q}\left(\sqrt{\frac{2E_b^{(k)}}{N_0}}\right) \tag{5.38}$$

Example 5.41

Assume that we are using a Wi-Fi-like OFDM technology with $N_a = 48$ data carriers out of a rester of N=64, and that our wireless channel is quasi-everywhere good like in Fig. 5.17, i.e., $H_k = 1$ on every subcarrier but a certain $k = \bar{k}$ where it is $H_{\bar{k}} = 0$ – a notch frequency. Assume also the SNR $E_s^{(k)}/N_0$ on the good carriers ($k \neq \bar{k}$) is so good that for the same carriers $BER_k \simeq 0$. Of course, on the bad carrier ($k = \bar{k}$) the signal is completely blanked, so that $BER_{\bar{k}} = 1/2$. What happens to the average BER of the modem? Should we expect a relatively good performance, given that the channel is quasi-everywhere good? Let us check, counting only active data carriers:

$$BER = \frac{1}{48}\sum_{k=0}^{47} BER_k = \sum_{k\neq\bar{k}} BER_k + BER_{\bar{k}} \simeq 0 + \frac{1}{48}\cdot\frac{1}{2} \simeq 10^{-2} \tag{5.39}$$

The results is highly unsatisfactory: with a channel that's almost perfect, the link BER is a meager 10^{-2}, practically useless for any application.

The example above tells us that the presence of deep notches in the frequency response of the channel may create a *BER floor* in the OFDM receiver, even if the SNR is on average very good. The solution to this lies in the many techniques for channel coding that we have already encountered in Chapter 3, and that become an essential feature of the technology – so much that OFDM is often addressed as COFDM (Coded OFDM).

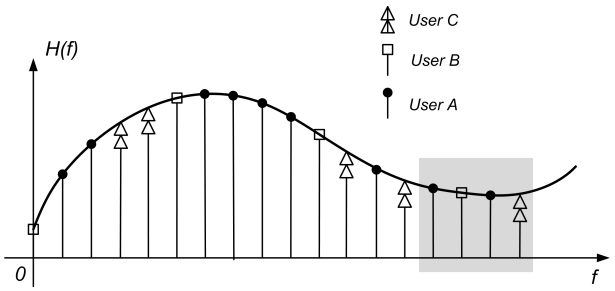


Figure 5.18 Allocation of subcarriers to different users in OFDMA

5.5 Further Features of OFDM Technologies

5.5.1 Flexible Multiplexing/Multiple Access

5.5.1.1 Multiuser OFDM (aka OFDMA) In chapter 2, we have already anticipated how to do multiplexing of digital signals via orthogonal signaling. Contrarily to its very name, in this chapter, OFDM has been introduced and developed not for multiplexing, but just to counteract in a single-user link the issue of frequency selectivity. This applies for sure to the many examples we did for Wi-Fi or DVB-T, but when it comes to 4G LTE, OFDM is used in a slightly different way, that we could label "Multiuser OFDM". In fact, in the downlink of LTE, the available data subcarriers are partitioned across the different users in the cell, so that each subcarrier is allocated to a different user. This recalls very much FDMA, but with a difference: in LTE, the subcarriers belonging to a certain user are not necessarily adjacent in frequency, i.e, they do not cover a connected or compact portion of the spectrum (and this would just be FDMA implemented digitally...). They are in general *scattered* across the whole occupied bandwidth like in Fig. 5.18.

This arrangement, that is currently called OFDMA (Orthogonal Frequency Division Multiple Access) even if it is not properly speaking multiple access, but just multiplexing, gives maximum allocation flexibility of the available capacity, since different users can be allocated a different number of subcarriers, thus experiencing a larger of smaller capacity (see again Fig. 5.18). In addition, scattering the subcarriers of all users across the whole spectrum is a *fair* allocation – in the case of adjacent carriers, a certain user may have all his/her subcarriers allocated in an unfavorable region of the spectrum, like that shaded in grey in Fig. 5.18. Scattering the subcarriers ensures that all users will have the same mixture of "good" carriers and "bad"carriers.

The receiver in the mobile terminal will demodulate the whole raster of N subcarriers with a standard OFDM demodulator, but will only consider those allocated to her/him by the network controller.

5.5.1.2 True OFDMA Proper OFDMA can also be done, and is actually done, in the uplink of a cellular network by following the same reasoning as in the dowlink, with the difference that the implementation is now distributed across user terminals, and access takes truly place from multiple points. Each user builds its own OFDM signal of order N but with a few active data carriers only, those that have been allocated by the network controller to that user, and treats all of the other carriers as virtual. The actual way this is done in LTE is called SC-FDMA and is a variant of "pure" OFDMA. We will deal with this later on.

5.5.2 Drawbacks of OFDM Technologies

Until now, all of the features of the new technology has been described as positive innovation, overlooking a bit some negative aspects that have to be taken into account.

5.5.2.1 Time-Selectivity Creating the multicarrier format implies lengthening the symbol time from the input value T_s to the OFDM symbol value $T_M = T_s \cdot N$ (still, we do not consider for simplicity virtual carriers and/or cyclic prefix). We do this on a massive scale (N up to several thousands) to counteract frequency selectivity. But, in our channel modeling we have always assumed stationarity of the channel, i.e., no channel variation across the symbol time. What symbol? Of course, the OFDM symbol. This may engender an issue when considering (TX/RX) mobility in the link. Take the value of the OFDM symbol T_M=896 μs for the DVB-T 8k mode, and assume (terrestrial television) $f_0 \simeq$800 MHz. If we do not want time selectivity problems, we have to make sure that the coherence time T_c channel is sufficiently long: $T_c > T_M$. Taking an intermediate factor of 20 in the definition of the coherence time (4.27), the condition of non-time-selectivity ends up in

$$v \leq \frac{c}{20 f_0 T_M} = 20.9 \text{ m/s} = 75.3 \text{ km/h} \tag{5.40}$$

so a low level of mobility is supported, otherwise the channel start being *doubly selective* – this is actually *not* an issue for TV broadcasting with fixed receiver. Examine on the contrary the parameters for 20-MHz 4G LTE cellular @ $f_0 = 2GHz$: N=2048 with N_v=847 virtual carriers, cyclic prefix N_G=160, subcarrier spacing f_{sc}=15 kHz, so that $T'_M = 1/f_{sc}$=66.7 μs. In this case, we find $v \leq +T_G$=112 m/s=405 km/h, capable of supporting connections to/from high-speed trains. In general, for wideband signals like those for which the adoption of OFDM makes sense, the issue of time selectivity is not a problem, because the length of the OFDM symbol T_M is still below the coherence time of the channel.

5.5.2.2 Sensitivity to Carrier Frequency Detuning Any I/Q demodulator for any signal format is affected in general by frequency detuning: the local reference frequency in the demodulator is not exactly equal the the RF frequency of the incoming signal. This may be due to the Doppler shift as explained in sect. 4.1.3, or simply by inaccuracy of the local oscillator. The result is that the I/Q received signal $y(t)$ is affected by a residual frequency offset term: $y(t) = x(t) \exp\{\jmath 2\pi \Delta f t\}$ where $1\Delta f$ is just the frequency detuning. What is the effect of this detuning in the data demodulator? If only switch on saubcarrier # k, we consider the receiver for the k-th carrier in Fig. 5.6 and we neglect noise we have

$$y_k(t) = c_0^{(k)} e^{\jmath 2\pi(\Delta f + k/T_M)t}$$

so that

$$\begin{aligned} z_0^{(k)} &= \frac{1}{T_M} \int_0^{T_M} y_k(t) e^{-\jmath 2\pi k t/T_M} dt = \frac{1}{T_M} c_0^{(k)} \int_0^{T_M} e^{-\jmath 2\pi \Delta f t} dt \\ &= c_0^{(k)} \frac{e^{\jmath 2\pi \Delta f T_M} - 1}{2\pi \Delta f T_M} = \jmath c_0^{(k)} e^{\jmath \pi \Delta f T_M} \text{sinc}(\Delta f T_M) \end{aligned} \tag{5.41}$$

If we now switch on carrier # i, like in (5.10), we also see the resurgence of an interfering term of carrier i onto carrier k:

$$I_0^{(i,k)} = \frac{c_0^{(i)}}{T_M} \int_0^{T_M} e^{\jmath 2\pi[(i-k)/T_M + \Delta f]t} dt = c_0^{(i)} \frac{e^{j2\pi[(i-k) f_{sc} T_M + \Delta f]} - 1}{j2\pi[(i-k) f_{sc} T_M + \Delta f]}$$

$$= c_0^{(i)} e^{\jmath\pi(i-k+\Delta f T_M)} \text{sinc}(i-k+\Delta f T_M) \ , \ i \neq k \tag{5.42}$$

Looking at the two equations, we can see that i) we have an amplitude/phase term on $c_0^{(k)}$ that will be handled by the subsequent frequency-domain equalizer and will be part again of H_k; for the phase term, there will be no harm, but the amplitude term with the sinc function is in general < 1 and causes a degradation (decrease) of the SNR on the decision variable; ii) the degradation is further enhanced by the interference term (5.42) caused by detuning that corrupts orthogonality – carrier i (and all of the others, not shown in the computation but actually there, that sum up on all values of $i = 0, 1, ..., N-1$ $i \neq k$) will interfere on carrier k, with an interference growing with growing Δf.

To keep the degradation tolerable, we have to enforce that $\Delta f << 1/T_M$, so that the sinc function in (5.41) is practically equal to 1, and the sinc function(s) in (5.42) is practically equal to 0. This clearly shows that the necessary accuracy in compensating possible frequency detunings has to be a small fraction of the subcarrier spacing $1/T_M$. This is much different form what happens in a conventional data demodulator for single-carrier modulation wherein (assuming NRZ pulse) the integration time of the matched filter will be T_s instead of T_M like in the present case, and so the necessary accuracy to keep the SNR degradation small will just be a small fraction of the symbol rate $1/T_s$ – the accuracy with OFDM has to be N times higher (the detuning N times smaller). Therefore, the carrier frequency estimation/correction function in the OFDM modem turns out to be *crucial*.

5.5.2.3 PAPR and SC-FDMA A final drawback of OFDM signaling comes from the properties of the multicarrier signal. For its very nature, the OFDM transmitted signal is a linear combination of a large number (N) of independent components, the baseband modulated data signals $y_k(t)$. For this reason, the statistics of the I/Q component of the transmitted signal $x_{OFDM}(t)$ tends to be Gaussian by virtue of the central limit theorem, and the amplitude of the signal will bear a Rayleigh pdf. This is conformed by simulation and measurements: contrarily to single-carrier modulations, the transmitted signal apparently loses its characteristic of carrying a digital data signal, and looks very much like an "analog" signal wherein the data symbol are not immediately visible (no data constellation can be displayed on the I/Q samples of the OFDM signal).

If the total power of the signal is P_{RF}, the pdf of the amplitude $a(t) = |x_{OFDM}(t)|$ of the signal is

$$f_A(a) = \frac{a}{P_{RF}} e^{-\frac{a^2}{2P_{RF}}} \ , \ a \geq 0 \tag{5.43}$$

and the pdf of the instantaneous signal power $p(t) = |x_{OFDM}(t)|^2/2$ is exponential:

$$f_P(p) = \frac{1}{P_{RF}} e^{-p/P_{RF}} \ , \ p \geq 0 \tag{5.44}$$

The average power is of course P_{RF}, but (at least in theory) the maximum (i.e., peak) instantaneous power is boundless. This cannot be clearly true for a physical signal, and is a pitfall of the central limit theorem. Conventionally, we call peak value P_{peak} that power level whose probability of being crossed is very low, say, 10^{-5}, and consider this value as the maximum that the signal can attain:

$$\int_{P_{peak}}^{\infty} \frac{1}{P_{RF}} e^{-p/P_{RF}} db = 10^{-5} \ \Rightarrow \ P_{peak} = 5 \ln 10 P_{RF} = 11.5 P_{RF} \tag{5.45}$$

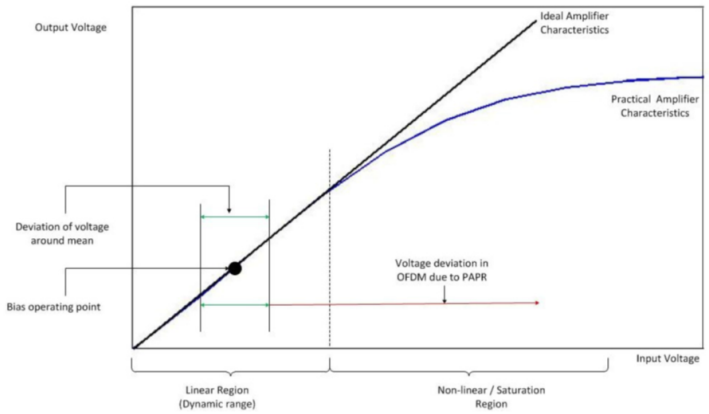


Figure 5.19 Power Amplifier and PAPR

From (5.45) we can say that the *peak to average power ratio* (what is usually called the PAPR) is 11.5 or 10.6 dB. This relatively high value of PAPR, much larger than with conventional modulations for which it is equal to 3-4 dB, creates power inefficiency in the transmitter – and this is a problem especially for portable terminals whose battery life has to be maximized.

In fact, if the PAPR of the signal is high, the power amplifier at radio frequency in the transmitter has to allow for a sufficient "margin" to be able to accommodate a signal that on average has a certain power P_{RF}, but occasionally goes up by more than 10 dB to $P_{RF} \cdot$ PAPR. This accommodation requires a higher-peak-power amplifier that wastes a considerable amount of supply power even when the instantaneous power of the signal is low, as is represented in Fig. 5.19. Here, we see the input-output characteristic of the RF amplifier, where we show the average operating point at P_{RF} and the necessary peak power value P_{peak}: the distance between the average power and the saturation (peak) power is hte PAPR and my be large. In such condition, the amplifier efficiency in terms of the ratio between the actual transmitted RF power and the supply power is low, and the battery wastes energy into unnecessary supply (that is eventually transformed into heat rather than RF power).

Many countermeasures to the PAPR have been investigated. The one that is applied into 4G LTE and 5G NR cellular for uplink transmission is the so-called SC-FDMA (Single-Carrier FDMA) format. We will not enter here into much details, we will only mention that the uplink signal of a mobile user is still generated by an OFDM modulator with the same number of carriers as in the downlink, with the usual pilot subcarriers and cyclic prefix, so that channel estimation/equalization can be performed and is effective the same way as before, including robustness towards frequency selectivity. In SC-FDMA, the data symbols, before being processed by the IFFT algorithm (5.18) is *precoded* in such a way as to reduce the PAPR. The complex values that are output by the precoder are then mapped onto the different subcarriers allocated to that user as in OFDMA, but they are no longer constellation values (and this why the PAPR is reduced) – they are precoded. In the receiver, the reverse processing is adopted with inverse precoding following FFT demodulation.

What is the precoding that is adopted (in 4/5G called *transform precoding*)? If the user we are considering in the uplink has M allocated subcarriers, precoding will just amount to

the DFT of the set of M symbols that will be sent on the M subcarriers. Intuitively, since OFDM modulation is just an IDFT, what we have here is a cascade of DFT and IDFT, so we expect that we are restoring constellation symbols in the time domain, instead of placing them in the frequency domain on the different subcarriers as (pure) OFDM does. The reality is a bit more complicate since i) the precoding DFT is order-M, while the IFFT of OFDM of order-N, so the two do not exactly interchange, and ii) the subcarriers allocated to one user may not be contiguous.

Let us drop for the moment ii), and let us assume that we send out M symbols $c_0^{(n)}$, $n = 0, 1, ..., M-1$ in the OFDM symbol time $m = 0$. From now on, we will drop the subscript 0 for simplicity. The precoded symbols will be

$$C^{(k)} = \frac{1}{M} \sum_{n=0}^{M-1} c^{(n)} e^{-\jmath 2\pi nk/M} \quad , \quad k = 0, 1, ..., M-1$$

so that, assuming that the carrier allocated to this users are just the first M in the raster of N, the SC-FDMA signal. for $0 \le t < T_M$ will be

$$x_{SF}(t) = \sum_{k=0}^{M-1} C^{(k)} e^{\jmath 2\pi kt/T_M} = \frac{1}{M} \sum_{k=0}^{M-1} \sum_{n=0}^{M-1} c^{(n)} e^{-\jmath 2\pi nk/M} e^{\jmath 2\pi kt/T_M}$$

$$= \frac{1}{M} \sum_{n=0}^{M-1} c^{(n)} \sum_{k=0}^{M-1} e^{\jmath \frac{2\pi k}{T_M}\left(t - n\frac{T_M}{M}\right)} = \sum_{n=0}^{M-1} c^{(n)} \frac{1 - e^{\jmath \frac{2\pi M}{T_M}\left(t - n\frac{T_M}{M}\right)}}{M\left(1 - e^{\jmath 2\pi\left(t - n\frac{T_M}{M}\right)/T_M}\right)}$$

$$= \sum_{n=0}^{M-1} c^{(n)} e^{\jmath \frac{2\pi(M-1)}{2T_M}\left(t - n\frac{T_M}{M}\right)} \frac{\text{sinc}\left(M\left(t - n\frac{T_M}{M}\right)/T_M\right)}{\text{sinc}\left(\left(t - n\frac{T_M}{M}\right)/T_M\right)}$$

$$\simeq e^{\jmath \frac{2\pi(M-1)t}{2T_M}} \sum_{n=0}^{M-1} (-1)^n c^{(n)} \text{Dir}\left(\frac{1}{T_M}\left(t - n\frac{T_M}{M}\right), M\right) \qquad (5.46)$$

where $\text{Dir}(x, M)$ is the Dirichlet kernel function

$$\text{Dir}(x, M) \triangleq \frac{\sin(\pi M x)}{M \sin(\pi x)} = \frac{\text{sinc}(Mx)}{\text{sinc}(x)} \qquad (5.47)$$

represented in Fig. 5.20 – a repetition with alternation of sign of the of sinc pulse. In (5.47), since $0 \le t < T_M$, only the central portion of the pulse is actually experienced.

After this painful computation, we understand the name SC-FDMA: looking at (5.46), we see that the signal is made of M symbols associated to a Dir pulse each, with symbol $c^{(n)}$ being transmitted at time nT_M/M: the OFDM symbol time is partitioned into M slots, each one used for one of the M user symbols, that are transmitted one after the other in time within a block. The symbols are then placed onto the *unique* subcarrier at frequency $(M-1)/(2T_M)$, just midway between the frequencies 0 and $(M-1)/T_M$ allocated for the user subcarriers. The subcarrier is *one*, and this is why this signal is called *single-carrier*. Other users in the cell will utilize another sub-band, so that the access is substantially FDMA.

The reason why SC-FDMA decreases the PAPR is then clear: it is basically a conventional single carrier modulation, with low PAPR, generated through the use of an OFDM modulator that allows anyway to perform good channel equalization. The difference with

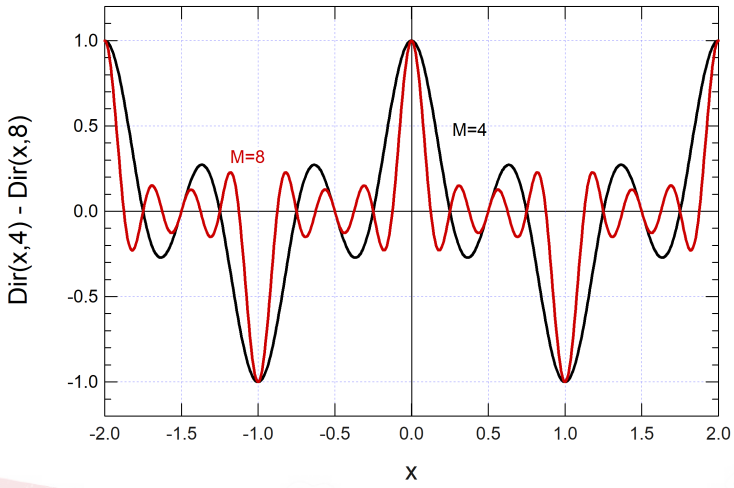


Figure 5.20 Dirichelet kernel function

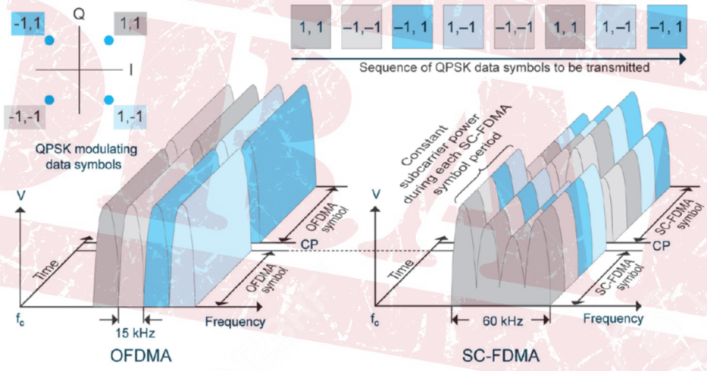


Figure 5.21 Difference between SC-FDMA and OFDM

conventional OFDM is that in the latter all data symbols are transmitted "in parallel" within the OFDM symbol time on different subcarriers. The symbols combine constructively or destructively according to the particular data pattern and may give either low or high values of signal amplitude – something that SC-FDMA does not do. This difference is finally sketched in Fig. 5.21. In 4G LTE, the subcarrier allocated at one user may not be contiguous. In that case, the representation that we have given of SC-FDMA still holds, the only difference being that the frequency spectrum of the signal is "scattered" across the overall 20-MHz channel bandwidth.

5.6 Multicarrier meets Information Theory: xDSL technologies and Shannon capacity of the colored Gaussian channel

The technology currently most widely used to provide "broadband" (i.e., high-capacity) services to residential subscribers, capable of efficiently and cost-effectively covering the

last mile of the access network, is the so-called xDSL family of digital standards for the subscriber network, where DSL stands for Digital Subscriber Line, and x can take on various values, characteristic of various systems with varying capacities: HDSL, ADSL, VDSL, in order of commercial availability. The following paragraphs will present the main characteristics of xDSL systems, and in particular, the calculation of the capacity of a colored Gaussian channel, as well as the DMT (Discrete Multi-Tone) technique for its maximization.

5.6.1 Architecture of a last-mile xDSL system

We have seen that the purpose of the access system for the fixed subscriber network is to connect the subscriber via a digital line to the service provider's Point of Presence (PoP) via a copper twisted pair connection is a broadband technology that allows data rates of between tens and hundreds of Mbps to be carried over twisted pair telephone lines, while simultaneously maintaining the traditional telephone service (POTS, Plain Old Telephone Service).

The most popular version of the xDSL family is ADSL, or Asymmetric DSL, for which the downstream capacity is greater than the upstream capacity. This asymmetry meets the typical needs of home users who download multimedia material from the network (i.e., video) generating much more traffic in the outgoing direction than in the return direction. The general architecture of the xDSL system is shown in Fig. 5.22. The subscriber has a digital connection to which a user terminal is connected, typically a WiFi router/access point, and an analog connection for a traditional telephone. Concentrating on the upstream flow, the two connections are coupled through a splitter, composed of a low-pass filter that selects the lower part of the spectrum for the analog connection and a high-pass filter that selects the upper part for the digital connection. The latter uses a remote ADSL transmission unit (ATU-R, ADSL Transmission Unit-Remote side, or simply the modem) that generates the modulated analog signal to be coupled with the analog telephone signal. The two signals are sent over a twisted pair to the exchange, where the telephone signal is decoupled (via a splitter on the exchange side) and sent to a traditional telephone network switch, while the modulated digital signal is demodulated by an ADSL transmission unit in the exchange (ATU-C, the exchange modem) and route towards the digital broadband network. Specifically, the user-specific flows from the various ATU-Cs are multiplexed by the DSLAM (Digital Subscriber Line Access Multiplexer), and forwarded to the optical-fiber transport network. Besides the exchange DSLAM, the only modification to the existing infrastructure consists of the ATU-R and ATU-C equipment to be installed at the subscriber site and in the exchange (or in an intermediate cabinet if the exchange is too far away), respectively.

5.6.2 Duplexing and Channel Modeling

Delivering a high-capacity digital stream over a transmission medium such as twisted pair poses some technical issues that are difficult to resolve. First, the coexistence of digital and analog lines must be solved. As already mentioned, the solution is a simple frequency-division multiplexing (FDM) depicted in Fig. 5.23: the telephone service occupies its own baseband, while the digital connection is modulated on a passband channel, spaced from the telephone band by a "guard" band. In addition, the digital connection is further duplexed using frequency-division duplexing (FDD), with the lower (narrower) band dedicated to the upstream and the upper (wider) band dedicated to the downstream.

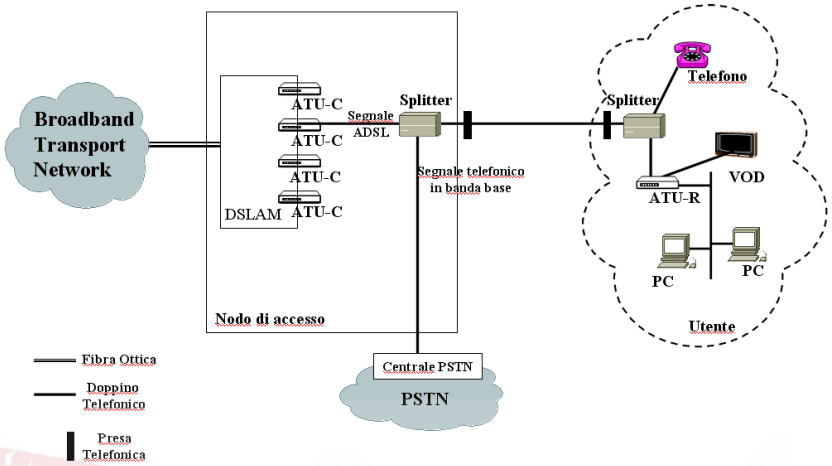


Figure 5.22 xDSL connecntion architecture

Figure 5.23 ADSL frequency plan

Once these details have been clarified, two strongly limiting issues still need to be considered, both related to the physical structure of the twisted pair: i) the frequency response of the medium, and ii) the noise to which it is subject. Regarding the frequency response calculation, the twisted pair can be modeled as a distributed-parameter transmission line whose input-output characteristics depend on its length, as well as on the type of twisted pair, for example, the cross-section of the cable. Over a typical length of one kilometer, the amplitude response is highly variable across the signal bandwidth, therefore a problem of channel frequency selectivity arises.

An approximate model of the amplitude response of the twisted pair as a function of the link length L is

$$|H(f;L)|_{dB} = -\alpha \cdot L \cdot \sqrt{f} \tag{5.48}$$

which, in addition to the usual exponential attenuation with the cable length, shows a rapid decay with frequency, and where α is a suitable constant that takes into account the physical characteristics of the cable, such as its cross-section. The channel is selective on the band of interest (selectivity occurs starting from a few kHz) and the signal is heavily distorted when considering a band of 1-2 MHz up to 12 MHz (!) for ADSL, ADSL2+, and VDSL services, respectively.

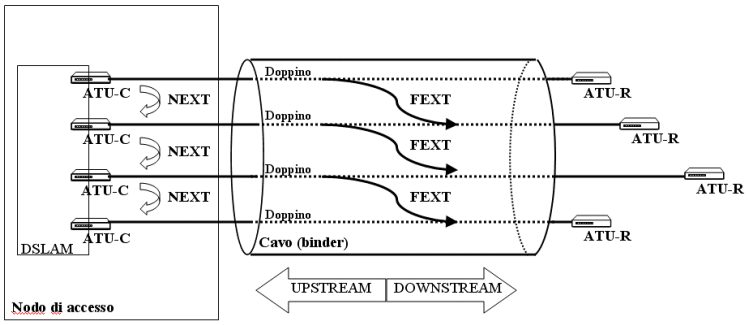


Figure 5.24 NEXT and FEXT in a xDSL binder

In addition to its (poor) frequency response, twisted pair cables are also subject to many interferences: what ultimately limits the performance of the connection is *crosstalk*. Crosstalk is the unwanted electromagnetic coupling between different pairs in the same cable. The twisted pair is twisted on itself to avoid such interference as much as possible, but it is not shielded from interference like a coaxial cable. Multiple unshielded twisted pairs are grouped together in the underground cable that connects the users to the cabinet or to the PoP, the so-called *binder* represented in Fig. 5.24. In the figure, we highlight two different crosstalk mechanisms: *far-end crosstalk* or FEXT, caused by transmitters *far* from the respective receiver, and *near-end crosstalk* or NEXT, caused by transmitters *close* to the receiver, as shown in Fig. NFEXT. The former is caused by signals traveling in the same direction as that of the reference pair, while the NEXT crosstalk arises from signals traveling in the opposite direction. To characterize crosstalk, we note that the various twisted pairs of a single cable (a dozen or more) generate *statistically independent* interferences (since they originate from independent communications). The sum of the various individual disturbances produces the total NEXT and FEXT, which (by virtue of the central limit theorem) can be modeled as *colored* (i.e., non-white) Gaussian noise processes, with a non-constant power spectral density. Coloring of the disturbance is related to the non-uniform spectral characteristics of the interfering signals, and is further enhanced by the selectivity of the (capacitive) coupling between conductors. Empirical analysis of the phenomenon resulted in a power spectral density of the NEXT as follows:

$$S_{NEXT}(f) \cong \alpha_{NEXT} \cdot f^{3/2} S_{INTER}(f) \tag{5.49}$$

where k_{NEXT} is a constant depending on the number of twisted pairs in the cable, and $S_{INTER}(f)$ is the power spectrum of the interfering signal. A similar model is also adopted for FEXT:

$$S_{FEXT}(f) \cong \alpha_{FEXT} \cdot f^2 \cdot L \cdot |H(f,L)|^2 S_{INTER}(f) \tag{5.50}$$

where L is the length of the coupling path and $H(f,L)$ is the frequency response measured between the two ends of the pair. The presence of this last component, which contains the attenuation term already discussed in (5.48), makes FEXT generally less harmful than NEXT. Notice that the interfered signal by FEXT (called the *victim* in ADSL jargon) is of the same order of magnitude as the interfering signal throughout the whole cable, while in NEXT the victim signal is weak (most of the interference occurs near the receiver, so that the victim has been attenuated by the twisted pair), whereas the interfering signal is strong (near the victim receiver, i.e., near the interfering transmitter).

Our discussion on crosstalk provides a better understanding of some aspects already examined. The upstream flow is highly susceptible to NEXT: in the PoP, the twisted pairs are collected in the binder immediately at the ATU-C output, and therefore, in the first part of the binder, the crosstalk of the (many) high-power downstream signals on the low-power upstream signals is very significant. To mitigate this phenomenon, the downstream is transmitted on the lower band of the frequency-division duplex, where the twisted pair attenuation is lower and where the NEXT is also lower, because the capacitive coupling between the twisted pairs is lower. In the downstream, the NEXT is reduced because in the first tens of meters of the connection, the user twisted pairs are *separated* and do not give rise to crosstalk – when they are inserted into the binder, the NEXT that is produced is reduced. We are now in a position to understand why the upstream bandwidth is narrower than the downstream bandwidth: widening it, would include additional frequency components that are attenuated and very much subject to NEXT, *not* leading to an appreciable increase in capacity, as we will see in more detail later on. This is why the service is inevitably *asymmetric*.

In addition to crosstalk, other typical twisted pair disturbances include impulsive noise and radio interference (unshielded twisted pair), as well as mismatches due to coupling transformers, and of course, receiver noise. All these phenomena, however, are (much) less significant than crosstalk. From the perspective of the communication channel, we have a problem of *distortion* and *additive Gaussian colored noise*, in a context in which achieving high bandwidth efficiency is essential. It is interesting to approach the issue from the perspective of information theory to derive Shannon capacity and, possibly, find a hint on how to implement a technology that approximates this capacity.

5.7 Shannon capacity of the colored Gaussian channel

In section refcapgausect we derived the Shannon-Hartley formula for the capacity of a Gaussian channel with uniform noise spectral density over a limited bandwidth B:

$$\mathcal{C} = \frac{1}{2T_s} \cdot \log_2 \left(1 + \frac{\sigma_X^2}{\sigma_W^2}\right) = B \cdot \log_2 (1 + SNR) = B \cdot \log_2 \left(1 + \frac{P}{N_0 B}\right) \text{ [bit/s]} \tag{5.51}$$

where SNR is the signal-to-noise ratio at the channel output, $P = \sigma_X^2$ denotes the power (variance) of the transmitted signal, and $S_w(f) = N_0/2$ is the two-sided power spectral density of the additive white Gaussian noise, so that $\sigma_W^2 = N_0/2 \cdot 2B = N_0 \cdot B$ is the power (variance) of the channel noise measured on the passband B. We intend to find how this result must be modified if i) the input signal is distorted by the communication channel, and ii) the noise is colored (ACGN, Additive Colored Gaussian Noise), that is, it has a power spectral density (d.s.p.) $S_W(f)$ that varies across the signal bandwidth. The first issue is easily solved. In fact, the situation just described, represented in Fig. 5.25 (a), is equivalent to that in Fig. 5.25 (b), in which the noise $W'(t)$ has a power spectral density

$$S'_W(f) = S_W(f)/|H(f)|^2 \tag{5.52}$$

and naturally retains Gaussian statistics. If the filter is invertible (as is the case for all physically realizable systems), the channel capacity calculated at the filter *output* is the same as that calculated at the filter *input*: an invertible transformation does not change mutual information. From a practical point of view, if we assume that we know the frequency response of the channel $H(f)$, we can add an inverse filter, i.e., an *equalizer* that perfectly compensates for channel distortion without changing capacity.

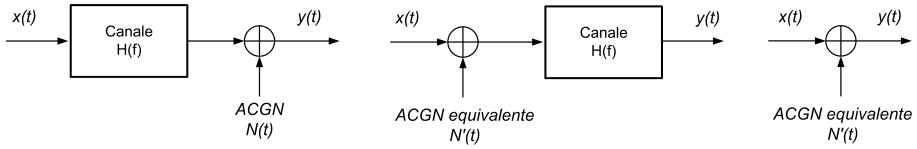


Figure 5.25 AWGN channel and its equivalent

Our equivalent problem is depicted in Fig. 5.25 (c), where the channel is not distorting, but the noise is colored with a power spectral density $S'_W(f)$. In particular, channel distortion has been "incorporated" into the psd of the equivalent colored noise $S'_W(f)$.

Now, the question arises as how to evaluate Shannon capacity of this new communication channel. The problem is solved by trying to reduce it to the already known case of Gaussian noise with flat power spectral density. Imagine splitting the entire available band B into a number N of equally spaced adjacent sub-bands, each with amplitude $\Delta f = B/N$. We can then further split our (single) digital information stream at rate R_b into N parallel sub-stream that we send out in parallel and independently of one another, using for each one a signal $x_k(t)$ with strictly bandwidth-limited bandpass modulation in the k-th sub-band, $[k\Delta f, (k+1)\Delta f)$, $k = 0, ..., N-1$. The separation of the sub-bands guarantees orthogonality of the various sub-channels: in practice, we are creating a multi-carrier transmission system similar to OFDM. If N is large, in each of these sub-bands the transmitted signal $x_k(t)$ "sees" a narrowband bandpass Gaussian noise $W'_k(t)$ with a practically flat power spectral density equal to

$$S'_{W_k}(f) \cong S'_W(k\Delta f) = S_W(k\Delta f)/|H(k\Delta f)|^2 \tag{5.53}$$

The diverse noises $W_k(t)$ on the various subchannels are white bandpass Gaussian processes with variable psd on non-overlapping bands. These processes are therefore uncorrelated and, since they are jointly Gaussian (since they are originated by filtering a common Gaussian wideband process), they are also *independent*. This ensures that the total capacity is given by the sum of the capacities of the individual (parallel) subchannels. The caacity of the kth subchannel is

$$\mathcal{C}_k = \Delta f \cdot \log_2 (1 + SNR_k) = \Delta f \cdot \log_2 \left(1 + \frac{P_k}{\sigma^2_{W'_k}}\right) =$$

$$\Delta f \cdot \log_2 \left(1 + \frac{P_k}{2 \cdot \Delta f \cdot S_{W'}(k\Delta f)}\right) = \Delta f \cdot \log_2 \left(1 + \frac{2 \cdot \Delta f \cdot S_X(k\Delta f)}{2 \cdot \Delta f \cdot S_{W'}(k\Delta f)}\right)$$

$$= \Delta f \cdot \log_2 \left(1 + \frac{S_X(k\Delta f)|H(k\Delta f)|^2}{S_W(k\Delta f)}\right) \text{ [bit/s]} \tag{5.54}$$

so that total Shannon capacity is

$$\mathcal{C} = \sum_{k=0}^{N-1} \mathcal{C}_k =$$

$$\sum_{k=0}^{N-1} \Delta f \cdot \log_2 \left(1 + \frac{P_k|H(k\Delta f)|^2}{2S_W(k\Delta f)\Delta f}\right)$$

$$= \sum_{k=0}^{N-1} \Delta f \cdot \log_2 \left(1 + \frac{S_X(k\Delta f)|H(k\Delta f)|^2}{S_W(k\Delta f)}\right) \text{ [bit/s]} \tag{5.55}$$

where $S_X(f)$ is the psd of signal $X(t)$. If the number of subbands is very high, Δf tends to 0, and the capacitance tends therefore to

$$\mathcal{C}_{ACGN} = \int_0^B \log_2 \left(1 + \frac{S_x(f)|H(f)|^2}{S_W(f)}\right) df \tag{5.56}$$

This theoretical result generalizes the white noise capacity formula. In addition to its theoretical significance, it suggests the adoption of a multicarrier technology to meet capacity as much as possible.

To sum up, let us recap (5.56), or its "discrete" version (5.54):

$$\mathcal{C} = \sum_{k=0}^{N-1} \mathcal{C}_k = \Delta f \sum_{k=0}^{N-1} \log_2 (1 + SNR_k) = \Delta f \log_2 \prod_{k=0}^{N-1} (1 + SNR_k) \quad [\text{bit/s}] \tag{5.57}$$

If all $SNR_k \gg 1$, then $1 + SNR_k \cong SNR_k$ and

$$\mathcal{C} = \Delta f \log_2 \prod_{k=0}^{N-1} SNR_k = \frac{B}{N} \log_2 \prod_{k=0}^{N-1} SNR_k =$$

$$B \cdot \log_2 \sqrt[N]{\prod_{k=0}^{N-1} SNR_k} = \log_2 SNR_{geo} \cong \log_2 (1 + SNR_{geo}) \tag{5.58}$$

where SNR_{geo} indicates the geometric mean of the signal-to-noise ratios on the various subcarriers. This shows that the capacity of the ACGN multicarrier system is that of a fictitious (single-carrier) AWGN system on the same total bandwidth and with a signal-to-noise ratio equal to SNR_{geo}. The geometric mean in dB is equivalent to the arithmetic mean of the values expressed in dB, so

$$\mathcal{C} \cong \frac{B \cdot (SNR_k|_{dB})_{aritm}}{3} [\text{bit/s}] \tag{5.59}$$

where $(SNR_k|_{dB})_{aritm}$ indicates the arithmetic mean of the SNR values expressed in dB on the individual sub-bands.

5.7.1 Maximizing the capacity of the colored Gaussian channel

The capacity formula of the Gaussian channel with colored noise (5.56) requires knowledge of the power spectrum $S_x(f)$ of signal $x(t)$. A further issue then arises: if there is no stringent constraint on the spectral distribution $S_x(f)$ of the signal $x(t)$, how can we optimally distribute (or more precisely *allocate*) the total signal power P_{tot} across the band B? From a practical point of view, considering the discrete, multi-carrier version of capacity (5.57), how can we determine the various P_k to be allocated to each sub-band so that the total capacity $\mathcal{C}$ of the link is maximized?

To simplify notation, let's denote $\sigma_k^2 \triangleq \sigma_{W_k'}^2$. The problem to be solved is then:
"Find

$$P_k \quad k = 0, ..., N-1$$

such that

$$\mathcal{C} = \max_{P_0,P_1,\ldots,P_{N-1}} \Delta f \sum_{k=0}^{N-1} \log_2\left(1+\frac{P_k}{\sigma_k^2}\right)$$

under the *constraint*

$$\sum_{k=0}^{N-1} P_k = P_{tot}$$

".

Without the maximum total power constraint, the problem is meaningless because capacity would grow indefinitely by increasing the powers P_k. Using a Lagrange multiplier λ, one must maximize

$$\Delta f \sum_{k=0}^{N-1} \log_2\left(1+\frac{P_k}{\sigma_k^2}\right) + \lambda\left(\sum_{k=0}^{N-1} P_k - P\right) \tag{5.60}$$

Differentiating wrt P_k and setting equal to 0 we get

$$\frac{\Delta f \log_2 e}{1+\frac{P_k}{\sigma_k^2}} \frac{1}{\sigma_k^2} + \lambda = 0 \quad , \quad k = 0,1,\ldots,N-1 \tag{5.61}$$

or

$$P_k + \sigma_k^2 = -\frac{\Delta f \log_2 e}{\lambda} \quad , \quad k = 0,1,\ldots,N-1 \tag{5.62}$$

If we now do the sum on k of all of the preceding equations we get

$$P_{tot} + \sigma_{tot}^2 = -N\frac{\Delta f \log_2 e}{\lambda} \quad \rightarrow \quad -\frac{\Delta f \log_2 e}{\lambda} = \frac{P_{tot}+\sigma_{tot}^2}{N} \tag{5.63}$$

where we have exploited the constraint $\sum P_k = P_{tot}$ and where $\sigma_{tot}^2 \triangleq \sum \sigma_k^2$ is the total noise power across all subbands. Using (5.63) in (5.62), we conclude that

$$P_k + \sigma_k^2 = \bar{P} \triangleq \frac{P_{tot}+\sigma_{tot}^2}{N} \tag{5.64}$$

In the continuous version (exercise for the reader) we get

$$S_x(f) + S_{W'}(f) = \bar{S} \quad , \quad \bar{S} \triangleq \frac{1}{B}\int_0^B \left(S_x(f) + S_{W'}(f)\right) df \tag{5.65}$$

which identifies the shape of the transmitted signal's power spectrum that maximizes the channel capacity with ACGN noise under the constraint of the fixed signal power P_{tot}.

5.7.2 The water-filling criterion

Let's first interpret the result regarding the particular shape of the transmitted signal power spectrum found by maximizing the ACGN channel capacity:

$$\begin{cases} S_x(f) = \bar{S} - S_{W'}(f) = \bar{S} - \frac{S_W(f)}{|H(f)|^2} & S_{W'}(f) < \bar{S} \\ S_x(f) = 0 & S_{W'}(f) \geq \bar{S} \end{cases} \tag{5.66}$$

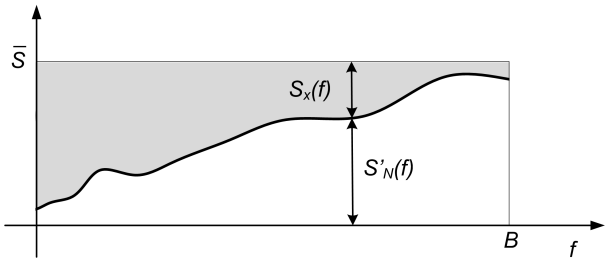


Figure 5.26 "Water Filling" criterion

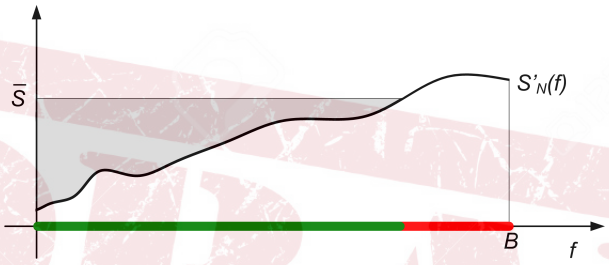


Figure 5.27 Noisy subbands in Water Filling

We see that such solution corresponds to a *constant* total spectral density, i.e., signal + noise, at the receiver equal to $S_{tot}(f) = S_x(f) + S_{W'}(f) = \bar{S}$, provided that the noise does psd not exceed the value $\bar{S}$ – this is called the *water-filling* solution. In fact, the situation is that represented in Fig. 5.26, and the particular shape of the transmitted spectrum that is obtained corresponds to what we get imagining fill with water an aquarium with a sandy bottom whose a variable height equals that of the modified noise spectrum $W'(t)$ – the resulting total psd represents the flat water level in the aquarium after filling up. The transmitted signal power P_{tot} is represented by the gray area in Fig. 5.26, i.e., by the "amount of water" poured into the aquarium, whilst the total noise power $P_{W'}$ is represented by the amount of sand on the bottom (white area in the figure).

The constant water level $\bar{S}$ (or $\bar{P}$ in its finite form) depends on the noise power $P_{W'}$ and on the available signal power P_{tot}. In some cases where the interference is particularly strong, an unfavorable situation may occur in certain areas of the spectrum (in xDSL this typically occurs at high frequencies, when the binder has twisted pairs with all users active and/or the victim twisted pair is very long). It may happen in fact that some frequency bands remain "uncovered", that is, the amount of water is not enough to cover the bottom of the aquarium, as in Fig. 5.27. The uncovered area corresponds to the one in which either the original noise spectrum is very strong, or the channel amplitude response is very small (and therefore the transmitted signal is greatly attenuated). Under these conditions, the water-filling theorem prescribes *not to use* such subbands (not wasting power there), and allocating the signal power only on the other sub-bands.

5.8 Discrete MultiTone Modulation

We reconsider now the problem of maximizing capacity in an ACGN channel, focusing on the practical implementation of the concepts just discussed. The capacity was calculated by dividing the available bandwdith B into N separate sub-bands, thus creating a multi-carrier transmission system. We then calculated the powers P_k to be associated with each sub-channel to maximize capacity. In this regard, we note that the water-filling criterion for allocating power to the sub-bands is counterintuitive: (5.64) says that $P_k = \bar{P} - \sigma_k^2$, that is, rather than trying to compensate for the noisy bands with more power, more power is given to the less noisy sub-bands. After power allocation, each sub-band operates at a specific SNR value, and these values will be very different from each other, resulting in very different capacities (typically very high at low frequencies and very low at higher frequencies). The capacity in each sub-band is

$$\mathcal{C}_k = \Delta f \log_2 \left(1 + \frac{P_k}{\sigma_k^2}\right) = \quad [\text{bit/s}] \tag{5.67}$$

Given a certain R_b that the modem intends to transmit on the link, we have reliable communication if we enforce that the bitrate $R_{b,k}$ on each subband is such that

$$R_{b,k} \leq \mathcal{C}_k \quad , \quad \sum_{k=0}^{N-1} R_{b,k} = R_b \tag{5.68}$$

We see that, after *power allocation*, we have also to perform the function of *bit-rate allocation* (or *bit allocation*) that is, partitioning the total bit rate among the various sub-bands.

In summary, the multicarrier technology used for xDSL formats, which is called DMT (Discrete MultiTone), maximizes system capacity by essentially performing two operations at the time of connection set-up: i) power allocation according to the water-filling principle, and ii) bit-rate allocation according to the capacity of the various channels. We already know how to implement a multicarrier system that is efficient from the point of view of modem architecture: we must implement FFT algorithms in transmission and reception. This is what is also done for xDSL, which also includes the cyclic prefix feature already examined for OFDM. The difference between OFDM and DMT lies in a fundamental factor: whenever a return channel is available from the receiver to the transmitter (this is not the case for instance in broadcasting networks) it is possible to make the modulation *adaptive* (Rate-Adaptive DMT), meaning that the bit rate is optimized as abovo with respect to the available capacity.

To achieve orthogonality of the various subchannels, we know that the symbol rate on each subcarrier must be the same, and must be equal to the spacing between the subcarriers: $R_M = \Delta f$. The only way to vary the bit rate on each subcarrier is therefore to change the COD/MOD on each subcarrier so that $R_{b,k} = \Delta f \cdot N_{b,k}$. In classic ADSL, Δf=4.3125 kHz and the coding rate r is kept *identical* on all subcarriers for simplicity – the bit rate is varied by changing the number of constellation point M_k: $R_{b,k} = \Delta f \cdot r \log_2(M_k)$.

Adapting the COD/MOD requires knowledge of the noise level on the subcarriers. When a connection is set-up, the transmitter in the modem sends a preamble consisting of uniform-power symbols, so that the receiver evaluates the signal-to-noise ratio on each subcarrier SNR_k, and sends these values back to the transmitter through the return channel. In such a way, the transmitter can allocate the right number of bits on each subcarrier,

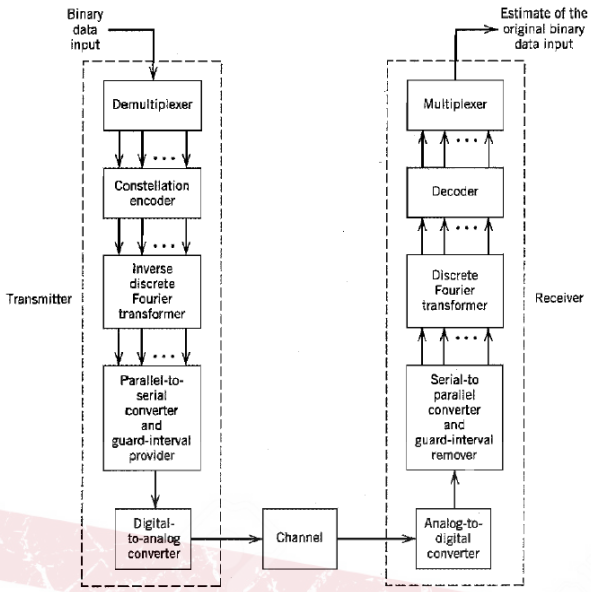


Figure 5.28 Architettura del link DMT

adaptively varying the constellation to be used. The most "noisy" subcarriers, i.e., those with low SNR_k, will use simple modulations with few bits/symbol (BPSK, QPSK), or will even be switched off according to the water-filling principle. Conversely, carriers with a high SNR_k will use a high-bit-per-symbol constellation. DMT modulation for ADSL provides a maximum of 32 upstream subcarriers with a cyclic prefix of 5 symbols, and 256 downstream subcarriers with a cyclic prefix of 32 symbols. The constellations can allocate up to a maximum of 15 bits per symbol, with trellis coding (a variant of convolutional coding). A Reed-Solomon code is also used as a packet error-correction code with prior interleaving/deinterleaving, as in the DVB-T coding scheme. The schematic of the ADSL transmission chain is therefore that of Fig. 5.28.

Unlike OFDM, where the mapping is unique across all subcarriers (so that the corresponding fucntion can be carried out before the S/P converter), the constellation encoder (or mapper) block in DMT maps the incoming parallel data bits into *different* QAM constellations according to the bit allocation algorithm. After mapping, the constellation symbols are scaled by different coefficients before undergoing IDFT (which transform the parallel data streams in the frequency domain into parallel data streams in the time domain), according to the power allocation algorithm (water filling) – the differences between DMT and OFDM is represented in Fig. 5.29. Note that for broadcast-type OFDM modulations, the power allocation operation would be meaningless anyway: to optimize capacity, one transmitter would be needed for each receiver (as in ADSL, where there is one ATU-C for each ATU-R), but this would go against the very definition of broadcasting.

Example 5.42

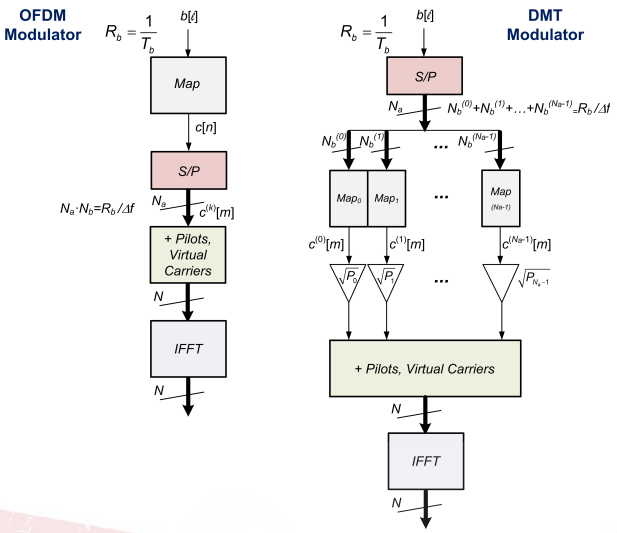


Figure 5.29 Confronto tra OFDM e DMT

The ACGN SHannon capacity is

$$\mathcal{C}_{ACGN} = \int_0^B \log_2 \left(1 + \frac{S_x(f)}{S_{W'}(f)}\right) df \tag{5.69}$$

Can we derive from this more general expression the simpler formula for the AWGN channel?

We can start by inserting $S'_W(f) = N_0/2$ into (5.69), but of course we cannot proceed further without specifying the shape of $S_x(f)$ – this can be only done by allocating the power of $X(t)$ according to the water-filling criterion:

$$S_x(f) = \bar{S} - N_0/2 \tag{5.70}$$

so that

$$\mathcal{C}_{ACGN} = \int_0^B \log_2 \left(\frac{2\bar{S}}{N_0}\right) df = B \log_2 \left(\frac{2\bar{S}}{N_0}\right) \tag{5.71}$$

On the other hand, we know that

$$\bar{S} = \frac{P_x + \sigma^2_{W'}}{2B} = \frac{P_x + N_0 B}{2B} \tag{5.72}$$

and finally

$$\mathcal{C}_{ACGN} = B \log_2 \left(\frac{2(P_x + N_0 B)}{2BN_0}\right) = B \log_2 \left(1 + \frac{P_x}{N_0 B}\right) \tag{5.73}$$

as we already knew.

CHAPTER 6

MANY IS BETTER THAN ONE - MIMO COMMUNICATIONS

"The Indian Goddess Durga"

—*Statue of the Warrior Goddess whose multiple arms help defeating the buffalo demon Mahishasura*

Having multiple arms give the Goddess supernatural powers. This Chapter's motto is in fact "Many is better than One" or, how to improve on conventional wireless communications by adding more antennas and transmitter/receiver pairs. The problem is, all of the antennas transmit at the same time, on the same band, and possibly with the same (spreading) code. Then what ? Let us discover the secrets of MIMO communications!

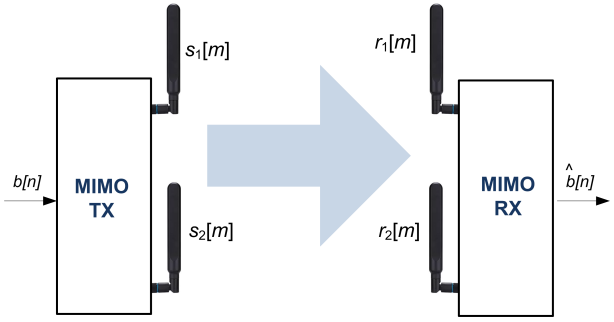


Figure 6.1 2×2 MIMO wireless communications setup

6.1 Introduction to Multiple-Input, Multiple-Output (MIMO) Communications

As Shannon taught us, the quest for efficiency in wireless communications is ever-lasting, and is limited by the capacity curve - a fundamental physical limit to achieve reliable communications. Once the SNR is known, there is no way to go beyond the celebrated valued $B \log_2(1 + \text{SNR})$ in terms of reliable bit-rate in a certain bandwidth B. In wireless communications, bandwidth and/or SNR can be substantially limited, so that reliable bit rate is limited as well. The idea of MIMO communications appear quite naive: if I do not make it with a single TX/RX link, let's add another one, like Fig. 6.1 suggests. The incoming data stream is split in two and sent to two separate modulators and TX antennas using the same bandwidth (on the same carrier), so that the total bit rate R_b is partitioned between the two. At the other end, two receiving antennas are used, and the two received stream are re-multiplexed at the receiver to eventually give the original stream. The first-rate impression is that the spectral efficiency is doubled since we are transmitting double as much as with a single 1×1 link on the same bandwidth. This is wrong, or at least too naive to be true, as Fig. 6.1 already shows: the TX antennas are usually simple omnidirectional aerials that transmit in any direction and are thus heard both by RX antenna 1 and antenna 2 at the same time. This creates interference between the two 1×1 links, to the detriment of reliability and ultimately decreasing actual capacity.

Before addressing this "difficult" problem, i.e., analyzing how much detrimental this cross-link interference is, and eventually understanding what a monster like the 8-antenna IEEE 802.11ax (Wi-Fi6) access point in Fig. can do, let us examine what can be done with simpler, 'heritage" multiantenna systems already in use in wireless communications before the 90's.

6.1.1 (Traditional) Receive Diversity Becomes Transmit Diversity

The idea of using multiple antennas to "improve" a wireless link dates back to the heroic times of radio technology. In particular, multiple antennas were used in the receiver to to improve on its sensitivity. Let us focus on the simple 1×2 systems with 2 receiving antennas depicted in Fig. . We assume that the transmitter sends out a constellation symbol s_m at time $t_m = mT_s$ where T_s is as usual the (MIMO) symbol interval. The receiver has two fully-equipped receivers with independent RX sections, so that, assuming narrowband communications on a frequency-flat channel, matched filtering reception and no ISI, the symbol-time outputs of the two matched filters in the two sections of the receivers are, as

Figure 6.2 Access Point a standard IEEE 802.11ax (Wi-Fi6) con 8 antenne

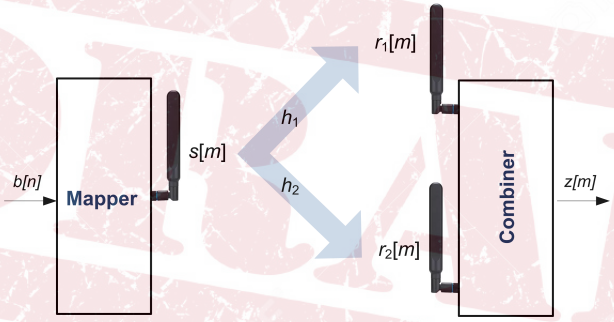


Figure 6.3 Diversity Reception

shown in Fig. 6.3,

$$r_m^{(1)} = h_1 s_m + w_m^{(1)} \quad , \quad r_m^{(2)} = h_2 s_m + w_m^{(2)} \tag{6.1}$$

The receiver has also a final processing unit that *combines* the two outputs above to give a final soft decision variable z_m on which constellation symbol estimation will be done. The two coefficients $h_1 = |h_1| \exp\{\jmath \angle h_1\}$ and $h_1 = |h_2| \exp\{\jmath \angle h_2\}$ just represent the amplitude and phase coefficients introduced by the frequency-flat channel on the transmitted symbol. Also, the two terms $w_m^{(1)}$ and $w_m^{(1)}$ are complex Gaussian noise values that turn out to be *statistically independent* since they come from different RF sections of the dual-antenna receiver. Since the two section are usually realized with identical technologies, we may assume that the variance of the two noise terms is the same - what is different is the SNR at the two antennas:

$$\text{SNR}_i = \frac{S_2 |h_i|^2}{2\sigma_w^2} \quad , \quad i = 1, 2 \tag{6.2}$$

where $S_2 \triangleq E\{|s_m|^2\}$ in the transmitted constellation power, and where σ_w^2 is the variance of the I and/or Q independent components of the noise terms.

6.1.2 Combining techniques

How should we combine $r_m^{(1)}$ and $r_m^{(2)}$ to get z_m? The simplest idea is to perform *selection combining*: form time to time, the receiver evaluates the SNR on the two antennas and *selects* the signal coming from the best, ignoring the other. This is very simple to implement, provided that some form of channel estimation can be done: to evaluate the $\text{SNR}_{1,2}$ we should theoretically know the values of $|h_{1,2}|$. In practice, this amounts to selecting the antenna whose output is larger, since the noise contribution on the two is the same. This strategy gives a link whose quality is of course batter than that of the 1×1 system, and also copes with possible variations in time of the channel(s), provided that selection is frequent enough wrt to the coherence time of the channel. In summary, $\text{SNR}_z = \max(\text{SNR}_1, \text{SNR}_2)$.

Can we do better that this? What is the optimum combining rule? Assume that we implement combining in the form of a linear combination of the antenna outputs with complex coefficients a_1 and a_2:

$$z_m = a_1 \cdot r_m^{(1)} + a_2 \cdot r_m^{(2)} \tag{6.3}$$

We may wish to find those values of the combination coefficients $a_{1,2}$ that gives maximum SNR. Considering (6.1), we have

$$\text{SNR} = \frac{S_2 |a_1 h_1 + a_2 h_2|^2}{2\sigma_w^2 (|a_1|^2 + |a_2|^2)} \tag{6.4}$$

From Cauchy-Schwarz inequality we know that

$$|a_1 h_1 + a_2 h_2|^2 \leq (|a_1|^2 + |a_2|^2) \cdot (|h_1^*|^2 + |h_2^*|^2)$$

where equality is attained when $a_{1,2} = c \cdot h_{1,2}^*$, $c \in \mathbb{R}$ a constant. Using both conditions, we get

$$\text{SNR}_{opt} = \frac{S_2 \cdot (|h_1|^2 + |h_2|^2)}{2\sigma_w^2} = \text{SNR}_1 + \text{SNR}_2 \tag{6.5}$$

The value of the constant c does not affect the SNR_{opt}, so that we may arbitrarily choose for simplicity $a_1 = h_1^*$ and $a_2 = h_2^*$. For instance, if $|h_1| = |h_2|$, then the improvement on the SNR wrt the conventional 1×1 link is a factor 2, i.e., 3 dB. Optimum combining is also called *maximal ratio combining* (MRC) for the property of maximizing the SNR after combination.

Example 6.43

We have just seen that the optimum value of the coefficients in maximal ratio combining is $a_{1,2} = c \cdot h_{1,2}^*$ where $c \in \mathbb{R}$ an irrelevant real-valued constant that may be set to 1 for convenience without affecting the received SNR. Let us now find that particular value of c that gives the minimum mean square error between the (soft) decision variable z_m and the constellation symbol s_m, so that optimum decision can be directly performed by a conventional constellation detector on z_m:

$$c_{opt} = \arg\max_c \text{E}\{|z_m - s_m|^2\} = \arg\max_c \text{E}\{|c(h_1^* r_m^{(1)} + h_2^* r_m^{(2)}) - s_m|^2\}$$

$$= \arg\max_c \text{E}\{|c[(|h_1|^2 + |h_2|^2)s_m + h_1^* w_m^{(1)} + h_2^* w_m^{(2)}] - s_m|^2\}$$

$$= \arg\max_c \text{E}\{|(\alpha c - 1)s_m + cW|^2\}$$

$$= \arg\max_c \mathrm{E}\{(\alpha c-1)^2|s_m|^2 + c^2|W|^2 + 2(\alpha c-1)c\Re[s_m W^*]\}$$

$$= \arg\max_c \{(\alpha c-1)^2 S_2 + c^2\alpha\cdot 2\sigma_w^2\} \tag{6.6}$$

Differentiating wrt c end equating to 0 we finally get

$$2\alpha(\alpha c-1)S_2 + 4c\sigma_w^2\alpha = 0 \quad\rightarrow\quad (\alpha c-1)S_2 + 2c\sigma_w^2 = 0$$

$$c_{opt} = \frac{1}{|h_1|^2+|h_2|^2+2\sigma_w^2/S_2} \tag{6.7}$$

so that the maximal-ratio coefficients that also minimize the mean-square error between the soft decision variable z_m and the transmitted constellation symbol s_m are

$$a_1 = \frac{h_1^*}{|h_1|^2+|h_2|^2+2\sigma_w^2/S_2} \;\;,\;\; a_2 = \frac{h_2^*}{|h_1|^2+|h_2|^2+2\sigma_w^2/S_2} \tag{6.8}$$

the denominator $|h_1|^2+|h_2|^2+2\sigma_w^2/S_2$ of the coefficient does optimum amplitude adjustment of the received signal so as to optimally match the "size" of the constellation, and also depends on the amount of noise as compared to the power S_2 of the transmitted constellation.

The selection and MRC/MMSE criteria can be extended easily to an arbitrary numberN_R of RX antenna - anice exercise for the reader. We conclude by remarking that the implementation of MRC calls for estimation of the channel coefficients $h_{1,2}$ (real/imaginary parts or amplitude/phase). A simpler yet less performing version of MRC just lets $a_i = \exp\{-j\angle h_i\}$ disregarding the channel amplitude (gain) - it is called *equal gain combining* and performs almost as fine as MRC when the power imbalance between the two channels is limited.

6.1.3 Diversity on the Rayleigh Channel

The beneficial effect of diversity, that we have up to now investigated for the Gaussian channel only, is even more striking if we consider a fading channel. Let us take into consideration in particular an order-N diversity receiver observing a single signal received through N "parallel" Rayleigh flat fading channels, and let us assume for simplicity that the fading on the different antennas are *statistically independent*. This is quite a crude approximation, but holds reasonably true when the distance between the diversity antennas is (substantially) larger that the wavelength of the radio signal. So the set of received signals at the N antennas at time mT_s is

$$r_m^{(i)} = h_i\cdot s_m + w_m^{(i)} \;\;,\;\; i=1,...,N \tag{6.9}$$

where each fading coefficient is normalized so as $\mathrm{E}\left\{|h_i|^2\right\}$=1 - we have on each antenna the same signal power as received from a unit-amplitude AWGN channel. Now we have to assume that the receiver has perfect knowledge of the channel gains (fading values) h_i, so that we can adopt maximal-ratio combining to yield the decision variable z_m as follows:

$$z_m = \sum_{i=1}^{N} h_i^* r_m^{(i)} \tag{6.10}$$

If we go on with the analysis, we find

$$z_m = s_m \sum_{i=1}^{N} |h_i|^2 + \sum_{i=1}^{N} h_i^* w_m^{(i)} = s_m \cdot \sum_{i=1}^{N} |h_i|^2 + W_m \tag{6.11}$$

Diversity reception has now *two* effects: on one hand, the usual reduction of the Gaussian noise variance since the noise components $w_m^{(i)}$ are mutually independent across antennas. On the other, we have an "averaging effect" of the signal fading due to combining and appearing in the term $\sum_{i=1}^{N} |h_i|^2$. This averaging makes low values of the fading amplitude less probable than in a one-antenna receiver, and improves a lot both the average BER and the outage probability.

Let us assume that s_m is a BPSK symbol, i.e., $s_m \in \{-1; 1\}$. In this case the normalized variance σ_w^2 of the I/Q noise components of $w_m^{(i)}$ is not $(2E_s/N_0)^{-1}$ as in the conventional one-antenna receiver (2.100). Rather, it is equal to $(2(E_s/N)/N_0)^{-1}$ since, with our normalization of the fading, the total received power from the N antennas is N times as much as in the conventional case. Therefore, the I/Q components of W_m in (6.11) both have a variance $\sigma_W^2 = \sigma_w^2 \sum_{i=1}^{N} |h_i|^2$. Under this condition, the value of the diversity BER for BPSK/QPSK *conditioned* on the *array* **h** of channel gains is easily found:

$$BER_D|\mathbf{h} = Q\left(\sqrt{\frac{2E_b}{N_0}\frac{\sum_{i=1}^{N}|h_i|^2}{N}}\right) \tag{6.12}$$

where E_b, as before for E_s, is the *total* energy-per-bit as received from all of the N antennas. To find the unconditional BER BER_D we are now to average $BER_D|\mathbf{h}$ over the (Gaussian) statistics of the array **h**. This is a bit complicated, so that we will only report the final result:

$$BER_D = \left[\frac{1}{2}\left(1 - \sqrt{\frac{E_b/N_0}{1+E_b/N_0}}\right)\right]^N \sum_{i=0}^{N-1} \binom{N-1+i}{i} \left[\frac{1}{2}\left(1 + \sqrt{\frac{E_b/N_0}{1+E_b/N_0}}\right)\right]^i \tag{6.13}$$

When E_b/N_0 is sufficiently large (say, larger than 10 dB), then

$$BER_D \simeq \binom{2N-1}{N} \left[\frac{1}{2}\left(1 - \sqrt{\frac{E_b/N_0}{1+E_b/N_0}}\right)\right]^N \tag{6.14}$$

We recall that for the conventional receiver with no diversity:

$$BER = \frac{1}{2}\left(1 - \sqrt{\frac{E_b/N_0}{1+E_b/N_0}}\right) \tag{6.15}$$

Figure 6.4 compares the BER curves of the different schemes discussed so far. The effect of diversity on the Rayleigh channel is apparent: the "native" BER of the one-antenna receiver improves exponentially with the diversity factor and the slope of the BER curve on a log scale like in Fig. 6.4 is changed accordingly. This improvement is called the *array gain* of the diversity receiver and is a typical behavior when Rayleigh fading is considered.

If we look back to (6.12), we can also guess what happens when the number of receiving antennas gets very large: by the law of large numbers the quantity $\sum_{i=1}^{N} |h_i|^2$ tends to its average value, which is equal to N. Therefore,

$$BER_D \to Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{6.16}$$

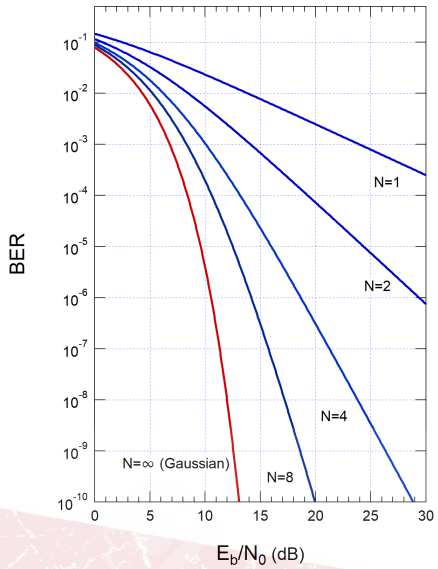


Figure 6.4 BER of the Diversity Receiver on the Rayleigh Channel(s)

so that the diversity Rayleigh channel is virtually turned into a Gaussian channel with no fading and with the same total received energy-per-bit E_b - a result that we will find again later on.

Example 6.44

What happens to the outage probability of the fading channel when we use diversity? As we did for the conventional receiver, we wish to find the *fraction of time* within which BER_D is satisfactory for our application, that is, $\text{BER}_D \leq \text{BER}_0$, where BER_0 is the application-dependent target value or *Quality-of-Service* (QoS). From (6.12), the probability that the link is not "satisfactory" (that is, we have an outage event) if

$$\Pr\{\text{BER} \geq \text{BER}_0\} = \Pr\left\{\sum_{i=1}^{N} |h_i|^2 \leq N/M\right\} \tag{6.17}$$

where we assume that BER_0 is the value we would get at the output of the un-faded AWGN channel ($|h_i| = 1 \forall i$) including the usual safety SNR margin M. The random variable $A \triangleq \sum_{i=1}^{N} |h_i|^2$ has a chi-squared distribution with $2N$ degrees of freedom, so that

$$\Pr\left\{\sum_{i=1}^{N} |h_i|^2 \leq N/M\right\} = \int_0^{N/M} \frac{1}{2^N(N-1)!}\alpha^{N-1}e^{-\alpha/2}d\alpha \tag{6.18}$$

The close-form expression of the integral is a bit cumbersome and is based on non-elementary functions, so that we resort to a Chernoff bound:

$$\Pr\{\text{BER} \geq \text{BER}_0\} \leq \left(\frac{\exp(1-1/M)}{M}\right)^N \tag{6.19}$$

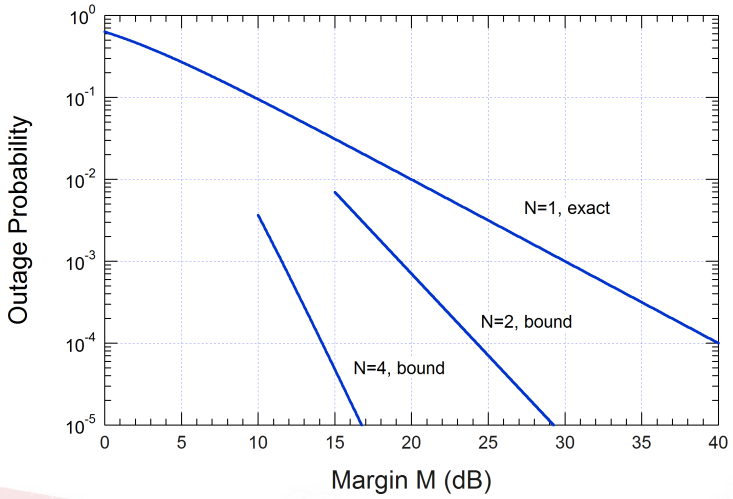


Figure 6.5 Outage Probability of the Diversity Receiver on the Rayleigh Channel(s)

We see that the outage probability decreases exponentially with N, as is shown in Fig. 6.5 as a function of the margin, and can be made arbitrarily small to meet a prescribed QoS requirement, at the expense of additional transmitted power. Of course, when $N \to \infty$ the outage probability is zero since there is no variability in the "instantaneous" BER: the channel is turned into AWGN and the BER is always equal to its average value.

6.1.4 Transmit Diversity

As already mentioned, diversity reception has been known and used for many years, even with analog communications, and represents an example of RX multi-antenna systems. More recently, the idea of using *transmit* diversity has emerged as a possibility strictly related to digital communications. The context is that of Fig. 6.6 (a): two different symbols from the same constellation $s_0^{(1)}$ and $s_0^{(2)}$ are transmitted at the same time $n = 0$ by the two TX antennas (1) and (2), and come at the (single-antenna) receiving end in a combination given by the two channel coefficients:

$$r_0 = h_1 s_0^{(1)} + h_2 s_0^{(2)} + w_0 \tag{6.20}$$

where w_0 is AWGN with variance σ_w^2 per component. There is of course no hope of recovering the two symbols separately from such received signal, nor there is any diversity here.

In diversity reception, the word *diverse* referred to the two receiving antennas that received two different (diverse) copies of the same TX signal. Here, we create diversity *in time* by replicating twice (i.e., transmitting again) the same pairs of symbols that have been previously transmitted by the two antennas, and observing a second version of the received signal. In particular, instead of just repeating $s_0^{(1)}$ and $s_0^{(2)}$ from antenna 1 and 2, respectively, as before, what we do is depicted in Fig. 6.6 (b), so that the received signal in

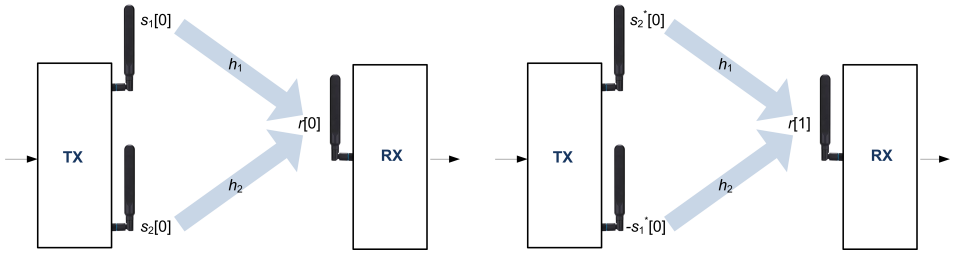


Figure 6.6 2×1 Transmit Diversity Setup

the second signaling time $n = 1$ will be

$$r_1 = -h_1 s^{*(2)}_0 + h_2 s^{*(1)}_0 + w_1 \tag{6.21}$$

where w_1 has the very same statistics as w_0. Diversity is found in time: we use two consecutive, diverse time intervals $n = 0$ and $n = 1$ to transmit two constellation symbols $s_0^{(1)}$ and $s_0^{(2)}$ - the usage of bandwidth is the same as in a conventional 1×1 system and as in receive diversity. We just use two antennas in transmission instead of two in reception.

What we have to find now is the kind of processing that we should use to retrieve the two symbols from the observation of r_0 and r_1: the idea is to do combination again, that is, deriving two soft decision variables, $z_0^{(1)}$ to decide on $s_0^{(1)}$ and $z_0^{(2)}$ for $s_0^{(2)}$, each obtained as a linear combination of the received values r_0 and r_1

$$z_0^{(1)} = a_{0,0} r_0 + a_{0,1} r_1^* \ , \ z_0^{(2)} = a_{1,0} r_0 + a_{1,1} r_1^* \tag{6.22}$$

where we have introduced a slight modification in the linear combination: since r_1 contains the complex-conjugate version of the constellation symbols, to create the soft decision variable we have used r_1^* instead of just r_1 to line-up with r_0. This said, we can take inspiration from the MRC solution of the previous section, and try to use $a_{0,0} = h_1^*$, and $a_{0,1} = h_2$ (since the coefficient of $s_0^{(1)}$ in r_1^* is h_2^*) in the combination to decide on $s_0^{(1)}$, disregarding for the moment the fact that each observed signal actually contains *both* symbols. Similarly we can try $a_{1,0} = h_2^*$, $a_{1,1} = -h_1$ for the combination relevant to $s_0^{(2)}$:

$$z_0^{(1)} = h_1^* r_0 + h_2 r_1^* \ , \ z_0^{(2)} = h_2^* r_0 - h_1 r_1^* \tag{6.23}$$

Using (6.20) and (6.21) in (6.23), we get

$$\begin{aligned} z_0 &= |h_1|^2 s_0^{(1)} + h_1^* h_2 s_0^{(2)} + h_1^* w_0 - h_1^* h_2 s_0^{(2)} + |h_2|^2 s_0^{(1)} + h_2 w_1^* \\ &= (|h_1|^2 + |h_2|^2) s_0^{(1)} + W_0 \end{aligned} \tag{6.24}$$

where we see the nice and unexpected property of this time-diversity transmission scheme, that is called after its inventor *Alamouti coding*: the cross-interference of symbol $s_0^{(2)}$ on $s_0^{(1)}$ has disappeared. Similarly and reciprocally, we easily find

$$z_1 = (|h_1|^2 + |h_2|^2) s_0^{(2)} + W_1 \tag{6.25}$$

with no interference of $s_0^{(1)}$ on $s_0^{(2)}$. In (6.24)-(6.25) the noise components W_0 and W_1 are still Gaussian (since they are linear combination of Gaussian variables) and both have variance $\sigma_W^2 = (|h_1|^2 + |h_2|^2)\sigma_w^2$ per component.

Our computation has shown that we can actually retrieve the two symbols with no mutual interference, even if in each received sample they are actually superimposed. We still have to understand what we gain with Alamouti coding. It is apparent that we do not gain anything wrt the conventional 1×1 link in terms of bit rate: we have the same two-symbols-in-two-transmission-times that we get with a single TX antenna. Let us compute the SNR on either z_0 or z_1 (call it SNR_{Ala}), we find

$$\text{SNR}_{Ala} = \frac{(|h_1|^2 + |h_2|^2)^2 S_2}{2\sigma_W^2} = \frac{(|h_1|^2 + |h_2|^2) S_2}{2\sigma_w^2} \tag{6.26}$$

i.e., the same result (6.2) that we got with conventional reception diversity.

Can we say that $\text{SNR}_{Ala} = \text{SNR}_1 + \text{SNR}_2$ just like in RX diversity? Yes, if we assume that the two TX antennas *both* transmit the same radio power as in a conventional 1×1 link - we have diversity gain up to 3 dB. No, if we assume (as is often the case) that it is the *total* RF power that is constrained by regulation, irrespective of it being radiated by 1,2, or many elements. In this case, a correct SNR comparison can be made between i) what we get with Alamouti coding (6.26) vs. ii) what we get with a single TX antenna using the same *total* transmitted RF power that we have with Alamouti. Under such hypothesis we have $SNR_1 = 2|h_1|^2 S_2/2\sigma_w^2$ and $SNR_2 = 2|h_2|^2 S_2/2\sigma_w^2$, so that in this case $\text{SNR}_{Ala} = (\text{SNR}_1 + \text{SNR}_2)/2$ and the diversity gain has vanished.

So what is the use of TX diversity? Consider the further case of a Rayleigh flat-fading channel as in , so that h_1 and h_2 are now unit-power Gaussian complex random variables. From (6.25) we see that we have the same situation as in (6.11) with $N = 2$. In conclusion, if the total TX power from the two antennas stays the same as in the 1×1 case, we still have an order-2 array gain just like in (6.12)-(6.14), so that the quality of the channel is improved by transmit diversity anyway.

6.2 General MIMO Links and Shannon Capacity

6.2.1 MIMO channel modeling and naive usage

After separately seeing what we can do with a double RX antenna first, and and a double TX antenna then, now we may wonder: what happens if we generalize to a proper MIMO 2×2 system like in Fig. 6.7 with two antennas on both ends ? The situation is now a bit more complicated in terms of channel description: as we can see in the figure, now we have *four* different wireless channels deriving from reception by any RX antenna of any TX antenna. This happens because the radiating elements normally used in MIMO are simple omnidirectional antennas, and are usually located, both at the TX and at the RX site, at a relatively small distance from each other. Assuming that we have frequency-flat radio channel(s), matched filtering, and ideal time synchronization, we end up with the following channel model at time mT_s:

$$\begin{cases} r_m^{(1)} = h_{1,1} s_m^{(1)} + h_{1,2} s_m^{(2)} + w_m^{(1)} \\ r_m^{(2)} = h_{2,1} s_m^{(1)} + h_{2,2} s_m^{(2)} + w_m^{(2)} \end{cases} \tag{6.27}$$

where the meaning of the various quantities is self-evident considering the previous sections. Once this is understood, we can think of further generalizing (6.27) to a generic $N_R \times N_T$

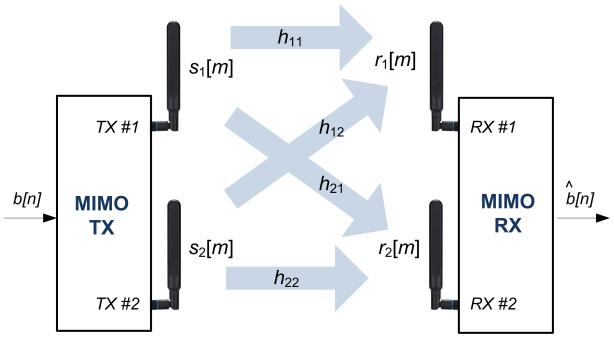


Figure 6.7 2×2 Full MIMO link

MIMO link with N_T transmitting antennas and N_R receiving antennas:

$$\begin{cases} r_m^{(1)} = h_{1,1}s_m^{(1)} + h_{1,2}s_m^{(2)} + ... + h_{1,N_T}s_m^{(N_T)} + w_m^{(1)} \\ \vdots \\ r_m^{(N_R)} = h_{N_R,1}s_m^{(1)} + h_{N_R,2}s_m^{(2)} + ... + h_{N_R,N_T}s_m^{(N_T)} + w_m^{(N_R)} \end{cases} \tag{6.28}$$

Casting (6.28) in a concise matrix form is straightforward:

$$\mathbf{r} = \mathbf{H}\mathbf{s} + \mathbf{w} \tag{6.29}$$

where we dropped for simplicity of notation the time-dependence on m,

$$\mathbf{r} \stackrel{\triangle}{=} \begin{bmatrix} r_m^{(1)} \\ \vdots \\ r_m^{(N_R)} \end{bmatrix}, \quad \mathbf{s} \stackrel{\triangle}{=} \begin{bmatrix} s_m^{(1)} \\ \vdots \\ s_m^{(N_T)} \end{bmatrix}, \quad \mathbf{w} \stackrel{\triangle}{=} \begin{bmatrix} w_m^{(1)} \\ \vdots \\ w_m^{(N_R)} \end{bmatrix} \tag{6.30}$$

and where

$$\mathbf{H} \stackrel{\triangle}{=} \begin{bmatrix} h_{1,1} & \dots & h_{1,N_T} \\ \vdots & \ddots & \vdots \\ h_{N_R,1} & \dots & h_{N_R,N_T} \end{bmatrix} \tag{6.31}$$

is the *MIMO channel matrix*. The noise vector $\mathbf{w}$ is zero-mean, Gaussian, and *spatially white*, that is, the different components of the vector are independent of each other, so that its covariance matrix is $\mathbf{C}_\mathbf{w} \stackrel{\triangle}{=} \mathrm{E}\{\mathbf{w}\mathbf{w}^\dagger\} = 2\sigma_w^2\mathbf{I}_{N_R}$, where $\mathbf{I}_{N_R}$ is the $N_R \times N_R$ identity matrix.

What is the consequence of (6.29) ? Apart from the noise vector $\mathbf{w}$, the transmitted symbol vector $\mathbf{s}$ is entangled in an apparently inextricable pattern of cross-interference terms. How can we recover the transmitted vector? Assume for simplicity that $N_R = N_T$; then, neglecting noise,

$$\mathbf{r} = \mathbf{H}\mathbf{s} \quad \rightarrow \quad \mathbf{s} = \mathbf{H}^{-1}\mathbf{r} \tag{6.32}$$

provided of course that the channel matrix be *invertible*. So the receiver, to recover the transmitted constellation symbols, has to i) know the channel matrix, to be able to ii) invert it, an apply it to the received vector. This operating mode of MIMO links is called *spatial*

multiplexing, just because we "multiplex" N_T streams (extracted in general from a single digital stream to be transmitted) onto *space*, that is, on different antenna resources or port (as they are called in MIMO parlance.) Multiplexing does *not* exploit here the usual resources of time, frequency, or code - the capacity of the link is augmented on the same band, time slot, or spreading code.

Can we say that we are really augmenting the total link capacity by a factor $N_R = N_T$? Let us take back into consideration the noise vector $\mathbf{w}$. In this case, we cannot say that $\mathbf{H}^{-1}\mathbf{r} = \mathbf{s}$, so that in general, applying $\mathbf{H}^{-1}$ to $\mathbf{r}$ we will get a soft-vector $\mathbf{z} = \mathbf{H}^{-1}\mathbf{r}$:

$$\mathbf{z} = \mathbf{H}^{-1}\mathbf{r} = \mathbf{H}^{-1}\left(\mathbf{H}\mathbf{s} + \mathbf{w}\right) = \mathbf{s} + \mathbf{H}^{-1}\mathbf{w} = \mathbf{s} + \mathbf{n} \tag{6.33}$$

The symbol vector is now accompanied by a noise vector $\mathbf{n}$ that is still Gaussian, but has different statistics wrt $\mathbf{w}$. In particular it is still zero-mean but its covariance matrix is now

$$\mathbf{C}_\mathbf{n} = \mathrm{E}\left\{\mathbf{n}\mathbf{n}^\dagger\right\} = \mathrm{E}\left\{\mathbf{H}^{-1}\mathbf{w}\mathbf{w}^\dagger\left(\mathbf{H}^{-1}\right)^\dagger\right\} = 2\sigma_w^2\left(\mathbf{H}^\dagger\mathbf{H}\right)^{-1} \tag{6.34}$$

We see from (6.34) that the variance of noise could be enhanced by MIMO processing, because inverting $\mathbf{H}^\dagger\mathbf{H}$ means multiplying by $\det(\mathbf{H}^\dagger\mathbf{H})^{-1} = 1/|\det(\mathbf{H})|^2$, so that the variance of the noise may *increase* when $|\det(\mathbf{H})|$ is small. It is true that the different components of the symbol vector $\mathbf{s}$ are recovered, but they may be affected by an overwhelming noise component, vanishing the purported capacity increase of spatial multiplexing - it is a matter of output SNR. How can we tackle and solve this problem? It is easy to understand that what we actually have to do is *computing the Shannon capacity of the MIMO link*.

Example 6.45

Let us focus on a simple 2×2 case with real-valued channel coefficients - deriving for instance form BPSK transmission with ideal coherent detection. It may happen in many cases that the geometric configuration of the TX and RX antenna systems is such that the distance r between antennas TX1-RX1 is the same as that between TX2-RX2 (this is because the distance d between TX1 and TX2 as well between RX1 and RX2 is much smaller than the distance r between TX and RX, as for a two-antenna Wi-Fi access point for indoor communications and a two-antenna laptop connected to the access point) - as a consequence, $h_{1,1} = h_{2,2}$. For the very same reason, $h_{1,2} = h_{2,1}$. In addition, $d \ll r$ also implies that $|h_{1,1}| \simeq |h_{2,1}|$ - there will be an arbitrary phase shift $\phi = 2\pi\Delta r/\lambda_0$ because of the small path diversity Δr between the two TX1-RX1 and TX1-RX2 paths. ($\lambda_0 = c/f_0$ is the carrier frequency wavelength, c the speed of light). The resulting channel matrix is *symmetric* and *persymmetric* (symmetric wrt the secondary diagonal):

$$\mathbf{H} = \begin{bmatrix} a & b \\ b & a \end{bmatrix} \tag{6.35}$$

If we had $|a| \gg |b|$ then the matrix would be substantially diagonal, and we would practically have two independent radio channels - it is easy to understand that in this case the total capacity of the radio link would be *doubled* wrt the conventional 1×1 arrangement. Unfortunately, this is not our "toy", where in $|a| \simeq |b|$, so that the cross-interference is very large and we may wonder if the capacity of the link is actually increased. In the *very bad* case wherein $a \simeq b$ then the channel matrix is *ill-conditioned*, its rank is basically 1, and its determinant is close to 0. What this means in terms of total capacity we still have to understand. In general,

$$\mathbf{H}^{-1} = \frac{1}{a^2 - b^2}\begin{bmatrix} a & -b \\ -b & a \end{bmatrix} \tag{6.36}$$

so that the variance of the I/Q noise components after detection is (on both antennas)

$$\sigma_n^2 = \sigma_w^2 \cdot (a^2 + b^2)/(a^2 - b^2)^2 \tag{6.37}$$

If the two values of $|a|$ and $|b|$ are close to each other, noise enhancement is intolerable. Take for example $a = 0.1$ and $b = 0.09$; then

$$\mathbf{H}^{-1} = \begin{bmatrix} 52.632 & -47.368 \\ -47.368 & 52.632 \end{bmatrix} \quad , \quad \sigma_n^2 = 5013.9 \cdot \sigma_w^2$$

$$\mathbf{z} = \begin{bmatrix} s^{(1)} + 52.632w^{(1)} - 47.368w^{(2)} \\ s^{(2)} - 47.368w^{(1)} + 52.632w^{(2)} \end{bmatrix} \tag{6.38}$$

Let us compare the SNR with and without MIMO. In the absence of TX antenna #2, the SNR at RX antenna #1 is $\text{SNR}_1 = a^2 S_2/2\sigma_w^2 = a^2\text{SNR}_n = 0.01\text{SNR}_n$ where SNR_n is a reference, channel-independent SNR $S_2/2\sigma_w^2$. Adding antenna TX2, the SNR on *both* antennas is, according to (6.37), equal to $\text{SNR} = (a^2 - b^2)^2/(a^2 + b^2)\text{SNR}_n = 2 \cdot 10^{-4}\text{SNR}_n = 0.02\text{SNR}_1$ with a degradation, for the same TX power on each antenna, of 17 dB !

Coming back to spatial multiplexing, it is pretty clear that when $N_R < N_T$ there's no hope of recovering the transmitted symbols from the sole observation of the received vector $\mathbf{r}$. But, before delving into Shannon capacity computation, we wish to address another practical case of interest, i.e., $N_R > N_T$. In this case, we may achieve spatial multiplexing as before, and add up to that also a "diversity effect" since the number of receive antennas is larger than the minimum required (N_T). The scenario $N_R > N_T$ is suggestive of uplink transmission towards a radio base station or an access point: the mobile device has a small number of transmitting antennas N_T, while the base station can have very many. The issue to be solved is that the detection problem is over-determined since we have more equations (N_R rows of the channel matrix or N_R receiving antennas) than unknowns (N_T transmitted symbols or N_T transmitting antennas), the $N_R \times N_T$ channel matrix is "portrait" and trivial matrix inversion does not apply.

The correct approach in this case is MMSE estimation: searching for a linear detector (i.e., a detection matrix $\mathbf{G}$ of size $N_T \times N_R$) to be applied to the $N_R \times 1$ received vector $\mathbf{r}$ to give the soft decision vector $\mathbf{z} = \mathbf{Gr}$, so that

$$\mathrm{E}\left\{\|\mathbf{z} - \mathbf{s}\|^2\right\} = \mathrm{E}\left\{\|\mathbf{Gr} - \mathbf{s}\|^2\right\} = \min \tag{6.39}$$

The solution to this problem is well known from statistical signal processing:

$$\mathbf{G} = \left(\frac{1}{\text{SNR}_n}\mathbf{I}_{N_T} + \mathbf{H}^\dagger\mathbf{H}\right)^{-1}\mathbf{H}^\dagger \tag{6.40}$$

When the SNR is very high, the optimum $\mathbf{G}$ is the so-called *Moore-Penrose pseudo-inverse* matrix $\left(\mathbf{H}^\dagger\mathbf{H}\right)^{-1}\mathbf{H}^\dagger$ that nulls the mutual interference between antennas (like inverting a square channel matrix) disregarding the effect of noise, whilst for low SNR the solution falls back to $\mathbf{H}^\dagger$ that represent a sort of "matched filtering" (or if you wish "maximal ratio combiner") to optimize wrt noise, disregarding mutual interference. Knowledge of the channel matrix is fundamental to devise any MIMO detector; in the case of the MMSE detector, knowledge of the SNR is also necessary - in (6.40) it is also necessary to estimate

the value of the *nominal* SNR $S_2/2\sigma_w^2$.

Example 6.46

Let us take back into consideration Example 45, and let us find the MMSE detector (6.40) for the same case. Let us also assume that SNR_n=40 dB, so that the SNR_1 is a good 20 dB, but the SNR after MIMO decoding is a meager 3 dB (SNR_1-17 dB due to the noise enhancement factor).

We have

$$\mathbf{H}^\dagger\mathbf{H} = \begin{bmatrix} 0.0181 & -0.018 \\ -0.018 & 0.0181 \end{bmatrix}$$

$$\frac{1}{\text{SNR}_n}\mathbf{I}_{N_T} + \mathbf{H}^\dagger\mathbf{H} = \begin{bmatrix} 0.018132 & -0.018 \\ -0.018 & 0.018132 \end{bmatrix}$$

$$\left(\frac{1}{\text{SNR}_n}\mathbf{I}_{N_T} + \mathbf{H}^\dagger\mathbf{H}\right)^{-1} = \begin{bmatrix} 3812.6 & 3784.8 \\ 3784.8 & 3812.6 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 40.61 & 35.36 \\ 35.36 & 40.61 \end{bmatrix}, \quad \mathbf{GH} = \begin{bmatrix} 0.8786 & -0.1189 \\ -0.1189 & 0.8786 \end{bmatrix} \tag{6.41}$$

so that finally

$$\mathbf{z} = \mathbf{Gr} = \mathbf{GHs} + \mathbf{Gw} =$$

$$\begin{bmatrix} 0.8786s^{(1)} - 0.1189s^{(2)} + 40.61w^{(1)} + 35.36w^{(2)} \\ 0.8786s^{(2)} - 0.1189s^{(1)} + 35.36w^{(1)} + 40.61w^{(2)} \end{bmatrix} \tag{6.42}$$

The SNR to be computed (the same on both components) is actually a SINR (Signal to Interference plus Noise Ratio) since there is a residual cross-interference which is *not* zero: the optimum MMSE matrix does *not* do exact inversion of the channel matrix - just minimizes the error power, *including* residual interference. The result is

$$\text{SINR}_z = \frac{(0.8786)^2 S_2}{(0.1189)^2 S_2 + ((40.61)^2 + (35.36)^2)\, 2\sigma_w^2} =$$

$$= \text{SNR}_n \cdot \frac{0.772}{0.0141 \cdot \text{SNR}_n + 2900} = 0.0003 \cdot \text{SNR}_n = 0.03 \cdot \text{SNR}_1 \tag{6.43}$$

Th SINR is better than in example 45 by 2 dB - not much, but this is due to the fact that the SNR_n is relatively high, and the MMSE detector is not much different from $\mathbf{H}^{-1}$.

Now, assume that we can add *two more receiving antennas* with propagation paths similar to the first two, so that

$$\mathbf{H} = \begin{bmatrix} 0.1 & -0.09 \\ -0.09 & 0.1 \\ 0.12 & -0.08 \\ -0.08 & 0.12 \end{bmatrix} \tag{6.44}$$

Doing some computation we get

$$\mathbf{H}^\dagger\mathbf{H} = \begin{bmatrix} 0.0389 & -0.0372 \\ -0.0372 & 0.0389 \end{bmatrix}$$

$$\frac{1}{\text{SNR}_n}\mathbf{I}_{N_T} + \mathbf{H}^\dagger\mathbf{H} = \begin{bmatrix} 0.038932 & -0.0372 \\ -0.0372 & 0.038932 \end{bmatrix}$$

$$\left(\frac{1}{\text{SNR}_n}\mathbf{I}_{N_T}+\mathbf{H}^\dagger\mathbf{H}\right)^{-1}=\begin{bmatrix}295.314 & 282.180\\ 282.180 & 295.314\end{bmatrix}$$

$$\mathbf{G}=\begin{bmatrix}4.135 & 1.640 & 12.86 & 10.24\\ 1.640 & 4.135 & 10.24 & 12.86\end{bmatrix}\ ,\ \mathbf{GH}=\begin{bmatrix}0.991 & -0.00892\\ -0.00892 & 0.991\end{bmatrix} \quad (6.45)$$

so that finally

$$\mathbf{z}=\mathbf{Gr}=\mathbf{GHs}+\mathbf{Gw}=$$
$$\begin{bmatrix}0.991s^{(1)}-0.00892s^{(2)}+4.135w^{(1)}+1.640w^{(2)}+12.86w^{(3)}+10.24w^{(4)}\\ 0.991s^{(2)}-0.00892s^{(1)}+1.640w^{(1)}+4.135w^{(2)}+10.24w^{(3)}+12.86w^{(4)}\end{bmatrix} \quad (6.46)$$

The SINR is now

$$\text{SINR}_z=\frac{(0.991)^2S_2}{(0.00892)^2S_2+((4.135)^2+(1.640)^2+(12.86)^2+(10.24)^2)\,2\sigma_w^2}=$$

$$=\text{SNR}_n\cdot\frac{0.982}{7.96\cdot10^{-5}\cdot\text{SNR}_n+290.0}=0.00336\cdot\text{SNR}_n=0.336\cdot SNR_1 \quad (6.47)$$

This is equivalent to a degradation wrt SNR_1 of 5 dB only, much better than the 17 dB or 15 dB degradation of the two-RX-antenna receiver - through MMSE detection and additional diversity gain we have realized a considerable improvement. The SNR in somewhat degraded, so we cannot afford very large constellations, but we have two communication "channels" to double the bit-rate.

What we have clearly understood from the previous section is that by just using up to N_T antennas in spatial multiplexing mode, we cannot pretend to multiply by N_T the capacity of the link. This would surely happen in the case of *independent* links with no mutual interference (diagonal channel matrix) but the general case of a full channel matrix **H** is more complicated. It would help if we could find an equivalent system configuration where by some trick the mutual interference between antennas be canceled. Then the total capacity, also in the strict Shannon sense, would be the sum of the partial capacities of the diverse (sub)channels - this is just the main subject of this Section.

6.2.2 MIMO Shannon Capacity

Finding the independent channels "hidden" into the MIMO matrix **H** can be done through a well-known algebraic tool called *Singular-Value Decomposition* (SVD) of a matrix. Let us focus on the general case $N_R > N_T$ that we have already addressed before. The SVD tells us that any complex-valued $N_R \times N_T$ matrix **H** can be decomposed as suggested in Fig. 6.8, that is, as the product of three special matrices:

$$\mathbf{H}=\mathbf{UDV}^\dagger \quad (6.48)$$

The special features of the matrices in (6.48) are the following: both the square matrices **U** (of size $N_R \times N_R$) and **V** (of size $N_T \times N_T$) are *unitary*, i.e., their inverse is equal to their own Hermitian: $\mathbf{U}^{-1}=\mathbf{U}^\dagger$, $\mathbf{U}^{-1}=\mathbf{U}^\dagger$. They describe a geometrical transformation that is akin to a rotation in Euclidean space, and does not alter mutual distances. This also means that if we apply a unitary matrix to a (spatially) white noise vector, the noise stays white:

$$\mathbf{n}=\mathbf{Uw}\rightarrow\mathbf{C_n}=\text{E}\left\{\mathbf{nn}^\dagger\right\}=\text{E}\left\{\mathbf{Uww}^\dagger\mathbf{U}^\dagger\right\}=2\sigma_w^2\mathbf{UIU}^\dagger$$

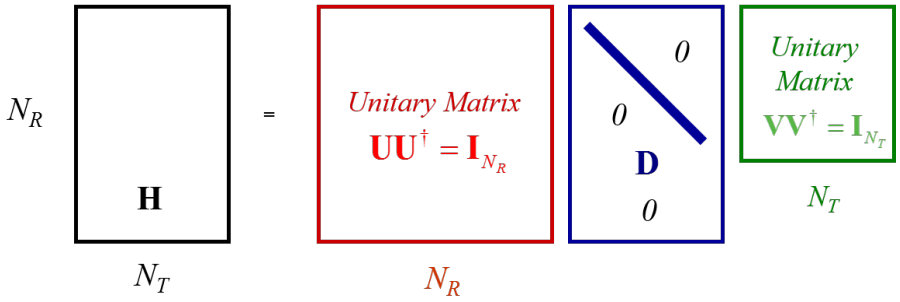


Figure 6.8 Representation of the SVD of channel matrix **H**

$$= 2\sigma_w^2 \mathbf{U}\mathbf{U}^{-1} = 2\sigma_w^2 \mathbf{I} = \mathbf{C_w} \tag{6.49}$$

In addition, matrix **D** is very special: i) it has the same $N_R \times N_T$ size as **H**; ii) its main $N_T \times N_T$ submatrix is diagonal, as shown in Fig. 6.8; iii) every other off-diagonal entry of **D** is 0, and and iv) the entries on the diagonal summarize the property and behavior of the original channel matrix **H**, since they are the square root of the N_T eigenvalues of the Hermitian, $N_T \times N_T$ positive-defined "squared" matrix $\mathbf{H}^\dagger\mathbf{H}$.

What has this to do with Shannon capacity? The SVD (6.48) gives us the opportunity to find an equivalent system to our original MIMO link (6.29), keeping in mind that any deterministic (known), invertible transformation on any signal does *not* change the Shannon capacity of the link. Using (6.48) in (6.29) we get

$$\mathbf{r} = \mathbf{U}\mathbf{D}\mathbf{V}^\dagger\mathbf{s} + \mathbf{w} = \mathbf{U}\mathbf{D}\bar{\mathbf{s}} + \mathbf{w} \tag{6.50}$$

where $\bar{\mathbf{s}} \stackrel{\triangle}{=} \mathbf{V}^\dagger\mathbf{s}$ is a "rotated" symbol vector with the same (second-order) statistics as **s** since the latter is (spatially) white (mutually independent symbols on the N_T transmitting antennas). Assuming that the channel matrix, as well as its SVD, is known to the receiver, we can process the received signal **r** with the matrix $\mathbf{U}^\dagger$ obtaining

$$\bar{\mathbf{r}} \stackrel{\triangle}{=} \mathbf{U}^\dagger\mathbf{r} = \mathbf{U}^\dagger\mathbf{U}\mathbf{D}\bar{\mathbf{s}} + \mathbf{U}^\dagger\mathbf{w} = \mathbf{D}\bar{\mathbf{s}} + \bar{\mathbf{w}} \tag{6.51}$$

where $\bar{\mathbf{w}}$ is a Gaussian vector with the same statistics as **w**. We have obtained a Shannon-equivalent link (capacity is the same because postprocessing with $\mathbf{U}^\dagger$ is invertible). If we imagine writing down explicitly the N_R equation corresponding to the matrix equation (6.51), and we just consider the first N_T only, we understand that such equations are *decoupled* (because **D** is in that range diagonal), and they describe N_T *parallel, independent Gaussian channels* with no mutual interference:

$$\bar{r}^{(i)} = d_i\bar{s}^{(i)} + \bar{w}^{(i)} \quad , \quad i = 1, ..., N_T \tag{6.52}$$

The channels are now statistically independent (since such are the components of $text\bar{b}fw$), so that the total capacity c_{MIMO} of the link is just the sum of the individual capacities c_i (4.93) of the N_T complex-valued parallel links (6.52):

$$c_{MIMO} = \sum_{i=1}^{N_T} c_i = \sum_{i=1}^{N_T} \log_2\left(1 + \mathrm{SNR}_i\right) = \sum_{i=1}^{N_T} \log_2\left(1 + \frac{d_i^2 S_2}{2\sigma_w^2}\right) \quad \text{bit/c.u.} \tag{6.53}$$

The meaning of (6.53) is pretty clear: with our MIMO link we have "opened" N_T parallel channels having *different* SNRs, therefore different capacities, according to the intrinsic gains of the channels d_i. If the channel matrix is ill-conditioned, one of the d_i's may be 0 and the capacity related to that channel goes to 0 as well. The actual multiplexing gain that can be obtained by the MIMO link in this spatial multiplexing mode is therefore strongly dependent on the values d_i and ultimately on the properties of the channel matrix (in addition of course of the nominal SNR of the link).

When $N_R < N_T$, the capacity is still formally given by (6.53), but the number of singular values, i.e., of parallel channels, is now N_R, and the summation runs from 1 to N_R - in the general case, from 1 to $\min(N_T, N_R)$. The value $\min(N_T, N_R)$ is the number of "virtual channels" contained in the MIMO link, and is representative of the multiplexing gain that we can get out of this technology - we can't expect to increase the bit rate beyond the minimum number of antennas available either at the TX or at the RX. But, increasing the number of receiving antennas N_R beyond the value N_T can be used to further increase capacity because of the improvement of the SNR due to *diversity* gain.

Example 6.47

Let us focus our attention on an unrealistic case of a 2×2 link with directive antennas that do not cause mutual interference, leading to a diagonal channel matrix with $h_{1,1} = h_{2,2} = 1$. The two SNRs are therefore $\text{SNR}_1 = \text{SNR}_2 = \text{SNR}_n$. We can compare the MIMO capacity with that of a 1×1 link: if the two MIMO TX antennas keep the same TX power as the single antenna, the MIMO capacity if just benefiting from a factor 2 of spatial multiplexing: $c_{MIMO} = 2\log_2(1 + \text{SNR}_n)$. More realistically, if we keep the same *total* TX power, we still have a spatial multiplexing factor equal to 2, but the baseline is this time the capacity of a single link with an SNR reduced by 3 dB because of the TX power split. Adopting MIMO seems to be advantageous only if $2\log_2(1 + \text{SNR}_n/2) > \log_2(1 + \text{SNR}_n)$, that is,... always, with increasing gain for increasing SNR.

What if we add two more (directive) RX antennas (to get to $N_R = 4$) so as RX3 only receives TX1, and RX4 only receives TX2? The channel matrix will be

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The reader may find on her/his own the SVD of this matrix. The result is $d_1 = d_2 = \sqrt{2}$, so that, in addition to a multiplexing gain equal to 2 as above, we also have a *diversity* gain equal to $d_1^2 = d_2^2 = 2$ yielding a capacity, for the same total TX power, $c_{MIMO} = 2\log_2(1 + \text{SNR}_n)$.

A different approach is adding the two additional antennas at the *transmitter*, so that TX3 communicates with RX1 and TX4 with RX2 to get to a 2×4 link. The channel matrix would be

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

and the two singular values are again $d_1 = d_2 = \sqrt{2}$ as before. del caso precedente. The multiplexing gain is still 2, what about capacity? Assuming that the total TX power stays constant, we halve the previous two-antenna power and we get $c_{MIMO} = 2\log_2(1 + \text{SNR}_n/2)$ just like in the 2×2 case - the additional TX antennas bring no additional

multiplexing advantage. We will deal later on with the correct approach to the usage of many TX antennas.

In our conceptual Example **??**, we highlighted the relation between multiplexing and diversity gain when the channel matrix has high (full) rank. When the channel matrix is somewhat degraded, the MIMO capacity decreases because of the degradation of the multiplexing gain. Let us see this with an example.

Example 6.48

Let us find the Shannon capacity of the 2×2 MIMO link of Esempio 45 with

$$\mathbf{H} = \begin{bmatrix} 0.1 & -0.09 \\ -0.09 & 0.1 \end{bmatrix} \tag{6.54}$$

and with SNR_n=45 dB. Computing the SVD, we find

$$\mathbf{H} = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & -0.7071 \end{bmatrix} \begin{bmatrix} 0.01 & 0 \\ 0 & 0.19 \end{bmatrix} \begin{bmatrix} -0.7071 & -0.7071 \\ 0.7071 & -0.7071 \end{bmatrix} \tag{6.55}$$

(since **H** is square, **U**=**V**) and we see that the two eigenvalues, as expected, are of different order of magnitude, one being relatively close to 0. The MIMO Shannon capacity is

$$\begin{aligned} c_{MIMO} &= \log_2\left(1 + 10^{-4}\text{SNR}_n\right) + \log_2\left(1 + 0.0361\text{SNR}_n\right) \\ &= 2.057 + 10.16 = 12.22 \ \text{ bit/c.u.} \end{aligned} \tag{6.56}$$

and the different contribution to the total capacity of the two virtual channels is apparent. The unit of capacity is as always in *bit per channel use*, where "channel use" means a full act of MIMO parallel transmission.

Using no MIMO and just the TX1-to-RX1 antenna with the same TX power, we get

$$c_{1-to-1} = \log_2\left(1 + 0.01\text{SNR}_n\right) = 8.31 \ \text{ bit/c.u.} \tag{6.57}$$

and the advantage of MIMO, albeit with a degraded second channel, is apparent. If we *double* the TX power in the 1-to-1 link, assuming to concentrate into TX1 the whole power previously transmitted across the two TX antennas, we have instead

$$c_{1-to-1} = \log_2\left(1 + 0.02\text{SNR}_n\right) = 9.31 \ \text{ bit/c.u.} \tag{6.58}$$

This is probably fairer than (6.57), since usually it is the total RF power that is dictated or constrained by regulation. Still, MIMO shows superiority in spite of the degraded MIMO channel.

To conclude, let us also evaluate the Shannon capacity of the 2×4 link of the same esempio. Computing again the SVD, we find this time

$$\mathbf{U} = \begin{bmatrix} -0.4870 & -0.1715 & -0.8539 & 0.06549 \\ 0.4870 & -0.1715 & -0.3047 & -0.8003 \\ -0.5127 & -0.6860 & 0.4057 & -0.3194 \\ 0.5127 & -0.6860 & -0.1160 & 0.5031* \end{bmatrix} \quad , \quad \mathbf{V}^\dagger = \begin{bmatrix} -0.7071 & 0.7071 \\ -0.7071 & -0.7071 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 0.2759 & 0 \\ 0 & 0.04123 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.59}$$

The capacity is clearly increased because both eigenvalues are larger than in the 2×2 case:

$$\begin{aligned} c_{MIMO} &= \log_2 (1 + 0.0017\text{SNR}_n) + \log_2 (1 + 0.0761\text{SNR}_n) \\ &= 5.775 + 11.23 = 17.01 \ \text{ bit/c.u.} \end{aligned} \tag{6.60}$$

We can find another version of (6.53) observing that, being d_i^2 an eigenvalue of $\mathbf{H}^\dagger\mathbf{H}$, then $1 + d_i^2 S_2/2\sigma_w^2$ is an eigenvalue of the matrix $\mathbf{I}_{N_T} + S_2/2\sigma_w^2\mathbf{H}^\dagger\mathbf{H}$. We also recall that the product of all eigenvalues of a matrix is equal to the determinant of that matrix. Therefore, form (6.53),

$$c_{MIMO} = \log_2 \prod_{i=1}^{N_T} \left(1 + \frac{d_i^2 S_2}{2\sigma_w^2}\right) = \log_2 \det \left(\mathbf{I}_{N_T} + \frac{S_2}{2\sigma_w^2}\mathbf{H}^\dagger\mathbf{H}\right) \ \ bit/c.u. \tag{6.61}$$

This is a celebrated and simple formula that gives synthetically the capacity of the spatial-multiplexing MIMO channel that we intended to derive.

Considering that the (RF) transmitted power by each antenna is $S_2/2$, we can say that $S_2/2 = P_T/N_T$ where P_T is the *total* radiated signal power on the radio band. Therefore,

$$c_{MIMO} = \log_2 \det \left(\mathbf{I}_{N_T} + \frac{P_T}{N_T\sigma_w^2}\mathbf{H}^\dagger\mathbf{H}\right) \ \ bit/c.u. \tag{6.62}$$

6.2.3 Capacity with TX power allocation

If the channel matrix is known to the transmitter as well, we can do better than (6.62). First, we can actually *pre-code* the symbol vector $\mathbf{s}$ with matrix $\mathbf{V}$ that is known to the transmitter, and we can transmit on the channel the precoded symbols. Before precoding, we can also allocate the total TX power unevenly across the different components of the symbol vector $\mathbf{s}$ - we will see in a while the reason why. Overall, this means sending out the pre-coded, power-controlled vector

$$\mathbf{v} \triangleq \mathbf{VAs} \ \ , \ \ \mathbf{A} \triangleq \text{diag}\{a_1, ..., a_{N_T}\} \tag{6.63}$$

where $a_i = \sqrt{2P_i/S_2}$, $i = 1, ..., N_T$ are the different amplitudes of the components of the symbol vector. The receiver gets the vector

$$\mathbf{r} = \mathbf{Hv} + \mathbf{w} = \mathbf{UDV}^\dagger\mathbf{VAs} + \mathbf{w} = \mathbf{UDAs} + \mathbf{w} = \mathbf{UD_A s} + \mathbf{w} \tag{6.64}$$

and post-process this received vector with the detection matrix $\mathbf{U}^\dagger$ to get

$$\bar{\mathbf{r}} = \mathbf{D_A s} + \bar{\mathbf{w}} \tag{6.65}$$

where $\mathbf{D_A} = \text{diag}\{a_1 d_1, ..., a_{N_T} d_{N_T}\}$ is the diagonal matrix of the received amplitudes, also including the virtual channel gains d_i.

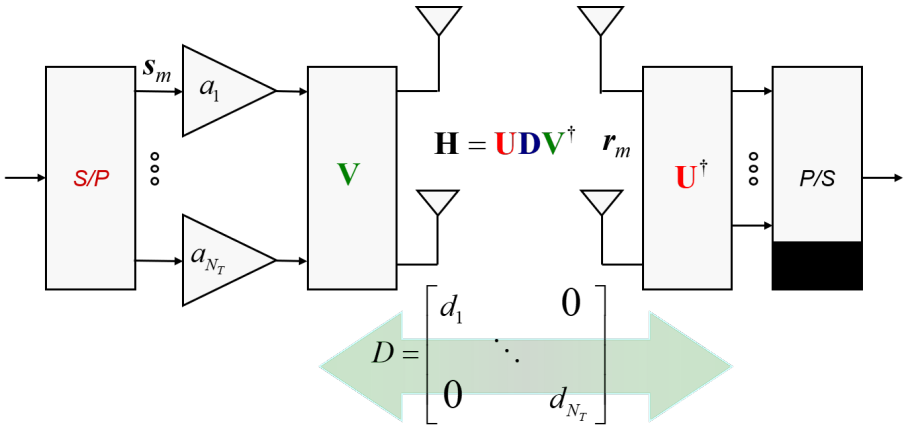


Figure 6.9 Pre- and Post-Processing with CSI at the TX

By virtue of this pre/post coding scheme summarized in Fig. 6.9, that calls for knowledge of Channel Status Information (CSI) both at the receiver and at the transmitter, we really experience at the RX the set of decoupled equations $\bar{r}_i = d_i a_i s^{(i)} + \bar{w}_i$, $i = 1, ..., N_T$ that we already encountered in our theoretical computation. In such condition, and the total capacity is

$$c_{MIMO} = \sum_{i=1}^{N_T} \log_2 \left(1 + \frac{d_i^2 a_i^2 S_2}{2\sigma_w^2}\right) = \sum_{i=1}^{N_T} \log_2 \left(1 + \frac{d_i^2 P_i}{\sigma_w^2}\right) \quad bit/c.u. \tag{6.66}$$

where, again, P_i is the (RF) transmitted power by antenna # i.

What is the use of power regulation or, as is more usually called, *power allocation* across the different transmitting antennas? The name allocation is more appropriate because what the transmitter does is allocating a total allowed power budget P_T across the antennas. This P_T is of course limited and is dictated by the specifications of a certain wireless communication standard - what we do is distributing this total power across the antennas within a total quantity: $P_1 + P_2 + ... + P_{N_T} = P_T$. This allocation is to be done according to a certain criterion, and in our spatial multiplexing context the criterion is: *find the allocation strategy that gives us the maximum Shannon capacity*. This is clearly a *constrained maximization* problem, where the constraint is the total power P_T to be spent:

$$\begin{cases} c_{MIMO}(P_1, ..., P_{N_T}) = \sum_{i=1}^{N_T} \log_2 \left(1 + \frac{P_i}{\sigma_w^2 / d_i^2}\right) = \max \\ P_1 + P_2 + ..., P_{N_T} - P_T = 0 \end{cases} \tag{6.67}$$

We show in the Appendix G that the solution to this problem is the so-called *Water-Filling* criterion, the very same as (5.66) that we encountered in the maximization of the capacity of the ACGN channel:

$$\begin{cases} P_i = \bar{P} - \sigma_i^2 \ , \ P_i > 0 \\ P_i = 0 \ \text{ elsewhere} \end{cases} \tag{6.68}$$

where

$$\sigma_i^2 \triangleq \frac{\sigma_w^2}{d_i^2} \ , \ \sigma_T^2 \triangleq \sum_{i=1}^{N_T} \sigma_i^2 \ , \ \bar{P} = \frac{P_T + \sigma_T^2}{N_T} \tag{6.69}$$

so that

$$c_{MIMO,wf} = \sum_{i=1}^{N_T} \log_2 \left(\frac{d_i^2 \bar{P}}{\sigma_w^2} \right) \tag{6.70}$$

In the WF solution, the quantity $\sigma_{w_i}^2$ is the equivalent noise variance on each independent MIMO channel, i.e., the receiver noise variance σ_w^2 scaled by the intrinsic channel power gain d_i^2. The WF criterion dictates that the sum between the allocated power P_i and the equivalent noise σ_i^2 has to be the same for each channel, and has to be equal to the value of the average signal-plus-noise total power $\bar{P}$ unless on that channel $\sigma_i^2 > \bar{P}$, in which case the antenna on that channel is *switched off* ($P_i = 0$).

The WF criterion is counterintuitive: seeing that that equivalent noise σ_i^2 on the different subchannels is variable, we would be tempted to allocate more power where σ_i^2 is larger, thus counterbalancing the intrinsic noisiness of the channel with a larger power, and having all subchannels working more or less at the same level of SNR. This is *not* what WF does. In fact, from (6.68) we see that at larger noise level corresponds smaller signal power - a difference-enhancing criterion. The idea is that, for maximizing the overall capacity, allocating (more) power on noisy antennas means wasting power with modest overall bit-rate increase. On the contrary, WF leverages on already good channels to make them better and allocating there most of the available reliable bit-rate (capacity).

Example 6.49

How much do we gain with WF wrt Esempio 48? Let us assume that we keep the same amount of total TX power $P_T = 2S_2$ as before or, in other words, $a_1^2 + a_2^2 = 2$. Computing the solution , we find

$$\sigma_T^2 = \left(\frac{1}{d_1^2} + \frac{1}{d_2^2} \right) \sigma_w^2 = 10,028\sigma_w^2 \quad , \quad \bar{P} = \frac{2S_2 + 10,028\sigma_w^2}{2} = S_2 + 5,014\sigma_w^2 \tag{6.71}$$

and the allocated power on each antenna is

$$\begin{cases} P_1 = \bar{P} - \sigma_1^2 = S_2 + 5,014\sigma_w^2 - 10,000\sigma_w^2 = S_2 - 4,986\sigma_w^2 \\ P_2 = \bar{P} - \sigma_2^2 = S_2 + 5,014\sigma_w^2 - 27.7\sigma_w^2 = S_2 + 4,986\sigma_w^2 \end{cases}$$

Notice that, es expected, $P_2 > P_1$ since $d_2^2 > d_1^2$. In addition, this is the correct solution assuming that $P_1 \geq 0$, otherwise we have to switch off antenna 1. In our case, $\text{SNR}_n = S_2/(2\sigma_w^2) = 45dB$ so that $S_2 = 63,250\sigma_w^2$ and $P_1 > 0$.

Finally, from (6.70)

$$c_{MIMO,wf} = \log_2 \left(\frac{d_1^2 \bar{P}}{\sigma_w^2} \right) + \log_2 \left(\frac{d_2^2 \bar{P}}{\sigma_w^2} \right)$$

$$= \log_2 \left(0.5 + 2 \cdot 10^{-4} \text{SNR}_n \right) + \log_2 \left(181 + 0.0722 \text{SNR}_n \right)$$

$$= 2.77 + 11.27 = 14.04 \text{ bit/c.u.} \tag{6.72}$$

and we gain about 2 bits/c.u. wrt to the case with no water filling. The gain would have been higher if the two virtual channels were more *unbalanced* than this.

At the end of the power allocation procedure, the virtual channels are left with considerably different SNRs, thus considerably different bit-rates to be allocated. This can be

theoretically done by using different MOD/COD formats on the different antennas, according to the specific Shannon capacity of that channel/antenna.

In the practice, providing the TX with full CSI is problematic, since the RX should feed back so many parameters (coefficients) to the TX, that the return channel risks being overloaded with *control* data not belonging to the data stream of the link. For this reason, most MIMO standards (e.g., 4G/5G) yes implement simplified pre-coding strategies that are based on a pre-assigned set of matrices established in the standard, that are chosen on the basis of the (very simple and concise ...) indication by the RX.

6.2.4 Space-Time Coding

As in conventional SISO links, the correct way to meet the constraint dictated by the Shannon capacity c in terms of information bit rate of the link (i.e., $R_b \leq \mathcal{C} = c/T_s$) is selecting not only the constellation to be used, but also the channel code with a certain rate. The MIMO link gives a new opportunity to construct channel codes: in conventional channel coding the redundancy bits are usually interspersed (interleaved) in *time* to give the encoded stream. In this way, the output signaling stream has a higher clock (rate) than the input stream, and the ratio between the input and the output rate, respectively, is the code rate $r < 1$.

What we can do in MIMO is interspersing the redundancy (parity) bit of a code not (only) in time, but (also) across *antennas*, that is, in *space*, so that we speak in general of *space-time coding*. Just to make a conceptual example that is not used in the practice, we might use two transmitting antennas with a systematic r =1/2 channel code, and then send all of the information (source) bits on antenna #1, and all redundancy (parity) bits on antenna #2. This is a pure "space" code where redundancy is added by virtue of the presence of a second antenna, as compared to traditional pure "time" codes like the ones that we use in SISO links.

A Space-Time (ST) block code can be constructed by creating a block $\mathbf{A}$ of M consecutive vectors (in practice, a $N_T \times M$ matrix):

$$\text{TIME} \rightarrow$$

$$\mathbf{A} \triangleq [\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{M-1}] = \begin{bmatrix} a_0^{(1)} & a_1^{(1)} & \dots & a_{M-1}^{(1)} \\ a_0^{(2)} & a_1^{(2)} & \dots & a_{M-1}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_0^{(N_T)} & a_1^{(N_T)} & \dots & a_{M-1}^{(N_T)} \end{bmatrix} \downarrow \text{SPACE} \tag{6.73}$$

The block $\mathbf{A}$ contains $N_T \times M$ symbols possibly representing a mixture of information and redundancy. In pure spatial multiplexing mode, no redundancy is present and the overall code rate is 1. The redundant symbols can be distributed in any fashion, horizontally (in time) or vertically (in space), and the overall code rate is the ratio between the number of information symbols wrt the total symbol number $M \cdot N_T$ - it is the decrease in total information rate wrt to pure spatial multiplexing.

The best known and most used ST code by far is a generalization of the Alamouti code that we have already seen as a strategy for 1×2 transmit diversity. We can extend that

strategy to a 2×2 "proper" MIMO link by creating the following ST code:

$$\mathbf{A} = \begin{bmatrix} s_0^{(1)} & -s_0^{*(2)} \\ s_0^{(2)} & s_0^{*(1)} \end{bmatrix} \tag{6.74}$$

This is nothing new, since it is exactly what we already introduced in Sect. 6.1.4 and that we labeled "transmit diversity". Now we recognize that it is actually a Space-Time code with overall rate r =1/2. Observe that the two rows of the code block $\mathbf{A}$ are *orthogonal* - this is the property that allows to remove interference between the two TX antennas in the linear block decoder. For this reason, Alamouti coding is a so-called *Space-Time Orthogonal Block* (STOB) code.

The optimum Alamouti 2×2 decoder is a straightforward generalization of (6.23). Let us start by saying that for any ST code, the channel equation for a code block is

$$\mathbf{R} = \mathbf{HA} + \mathbf{W} \tag{6.75}$$

where, with self-evident notation, $\mathbf{R}$ and $\mathbf{W}$ are $N_R \times M$ matrices such that

$$\mathbf{R} \stackrel{\triangle}{=} [\mathbf{r}_0, \mathbf{r}_1, ..., \mathbf{r}_{M-1}] \quad , \quad \mathbf{W} \stackrel{\triangle}{=} [\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_{M-1}] \tag{6.76}$$

For the Alamouti code we have

$$\mathbf{R} = \begin{bmatrix} r_0^{(1)} & r_1^{(1)} \\ r_0^{(2)} & r_1^{(2)} \end{bmatrix} = \begin{bmatrix} h_{1,1}s_0^{(1)} + h_{1,2}s_0^{(2)} & -h_{1,1}s_0^{*(2)} + h_{1,2}s_0^{(*1)} \\ h_{2,1}s_0^{(1)} + h_{2,2}s_0^{(2)} & -h_{2,1}s_0^{*(2)} + h_{2,2}s_0^{(*1)} \end{bmatrix} + \mathbf{W} \tag{6.77}$$

We recall the 1×2 Alamouti detector (6.23) that we already know, and we update it in our current context assuming that we are observing antenna RX1 only:

$$z_0^{(1)} = h_{1,1}^* r_0^{(1)} + h_{1,2} r_1^{*(1)} \quad , \quad z_0^{(2)} = h_{1,2}^* r_0^{(1)} - h_{1,1} r_1^{*(1)} \tag{6.78}$$

We can cast this equation in matrix format as follows:

$$\mathbf{z} \stackrel{\triangle}{=} \begin{bmatrix} z_0^{(1)} \\ z_0^{(2)} \end{bmatrix} = \begin{bmatrix} r_0^{(1)} & -r_1^{*(1)} \end{bmatrix} \begin{bmatrix} h_{1,1}^* & h_{1,2}^* \\ -h_{1,2} & h_{1,1} \end{bmatrix} \tag{6.79}$$

With a second RX antenna we add *receive diversity* on top of the already present transmit diversity, so that to further capitalize on the former, we just combine the output of the two antennas as if coming from two separate decoders of the same code:

$$\mathbf{z} = \begin{bmatrix} r_0^{(1)} & -r_1^{*(1)} \end{bmatrix} \begin{bmatrix} h_{1,1}^* & h_{1,2}^* \\ -h_{1,2} & h_{1,1} \end{bmatrix} + \begin{bmatrix} r_0^{(2)} & -r_1^{*(2)} \end{bmatrix} \begin{bmatrix} h_{2,1}^* & h_{2,2}^* \\ -h_{2,2} & h_{2,1} \end{bmatrix} \tag{6.80}$$

This is the final decoding equation for the Alamouti 2×2 code. First, we can show that (6.80) represents the maximum-likelihood decoder of the code, as well as minimum mean-square error solution (in the Gaussian regime, the two criteria lead to the same result as far as a linear detector is concerned). Let us discuss a bit the gain that we can attain from this ST code. The analysis is a bit complicated since we should plug (6.77) into (6.80). Recalling (6.24)-(6.25) and (6.80), we find

$$z_0^{(1)} = (|h_{1,1}|^2 + |h_{1,2}|^2 + |h_{2,1}|^2 + |h_{2,2}|^2) s_0^{(1)} + W_1$$

$$z_0^{(2)} = (|h_{1,1}|^2 + |h_{1,2}|^2 + |h_{2,1}|^2 + |h_{2,2}|^2)s_0^{(2)} + W_2 \tag{6.81}$$

where W_1 and W_2 are Gaussian complex with variance per component equal to $\sigma_W^2 = (|h_{1,1}|^2 + |h_{1,2}|^2 + |h_{2,1}|^2 + |h_{2,2}|^2)\sigma_w^2$. Again, cross-interference between antennas is canceled, and the SNR is

$$\text{SNR}_{ALA} = \frac{S_2(|h_{1,1}|^2 + |h_{1,2}|^2 + |h_{2,1}|^2 + |h_{2,2}|^2)}{2\sigma_w^2} \tag{6.82}$$

Now, let us do the usual SNR analysis - we have now *four* 1×1 SNRs to compare with SNR_{ALA}. If the TX power is *individually* left unchanged at any antenna, then trivially $\text{SNR}_{ALA} = \text{SNR}_{1,1} + \text{SNR}_{1,2} + \text{SNR}_{2,1} + \text{SNR}_{2,2}$ with self-evident notation, and we have up to 6 dB gain. But, as remarked for the 1×2 code, if it is the *total* power that is left unchanged, then

$$\text{SNR}_{ALA} = \frac{\text{SNR}_{1,1} + \text{SNR}_{1,2}}{2} + \frac{\text{SNR}_{2,1} + \text{SNR}_{2,2}}{2} \tag{6.83}$$

We have here a factor-2 array gain coming from the two TX antennas (as in the 1×2 case) combined with a *factor-2 diversity gain* coming from the 2 RX antennas.

The reader may wish to generalize Alamouti coding to a generic $N_R \times 2$ MIMO link, derive the generalized form of the detector (6.80), and then the resulting generalized SNR_{ALA} (6.83). Unfortunately, no generalization of Alamouti coding for more than $N_T = 2$ TX antennas exists, due to the (proven) impossibility of finding STOB's with complex-valued codeblocks with $N_T > 2$.

In practical systems, ST coding is seen as a downgraded alternative wrt full spatial multiplexing, depending on the status of the channel. The assumption is that the receiver is able to i) evaluate the status of the channel, in particular the rank of the channel matrix, ii) decide on the most opportune operating mode of the MIMO encoder, and ii) send the information back to the transmitter. 4G and 5G cellular standards both support full spatial multiplexing and the diverse options of Alamouti coding.

6.2.5 MIMO OFDM

Let us now go back to the fundamentals, in particular to the main MIMO channel model (6.28), that is based on the fundamental assumption that each of the $N_R \times N_T$ radio channels in the link are frequency-flat, so that propagation can be just characterized by the (frequency-independent) amplitude /phase coefficient $h = |h|e^{j\angle h}$. How can this be possibly true considering modern wideband digital transmissions? In a more general model, each constant $h_{i,\ell}$ should be replace by an *impulse response* $h_{i,\ell}(t)$, and the product with the constellation symbol s_m^ℓ should the replaced by the *convolution* with the whole transmitted signal $s^\ell(t)$ from transmit antenna #ℓ to give the i-the received signal $r^{(i)}(t)$ at receiving antenna #i.

There is actually nothing wrong with the frequency-flat modeling that we have adopted and discussed until now, provided that we intend that it is applied to an OFDM (or in general multicarrier) signal with cyclic prefix as in (5.11). In this case, all MIMO equations that we have seen and used up to now are *relevant to a single subcarrier*, on which by definition the response of the channel is frequency-flat. Let us recall the main equation characterizing the received signal at OFDM time m on subcarrier k ina conventional SISO link:

$$z_m^{(k)} = c_m^{(k)} \cdot H_k + w_m^{(k)} \quad k = 0, 1, ..., N-1 \tag{6.84}$$

where N is the number of subcarriers, $c_m^{(k)}$ is the data symbol, $w_m^{(k)}$ is the noise component, and $H_k = H\left(\frac{k}{T_M}\right)$ is the frequency response of the wireless channel at the k-th subcarrier frequency k/T_M. If we generalize this to a 2×2 MIMO link , we get easily

$$z_m^{(1;k)} = c_m^{(1,k)} \cdot H_{1,1;k} + c_m^{(2,k)} \cdot H_{1,2;k} + w_m^{(1;k)}$$

$$z_m^{(2;k)} = c_m^{(1,k)} \cdot H_{2,1;k} + c_m^{(2,k)} \cdot H_{2,2;k} + w_m^{(2;k)} \tag{6.85}$$

where we have now of course four channel frequency responses, corresponding to the four wireless channels between the two pairs of antennas. The equations are the same as those in (6.28) with an index more: the subcarrier number k. Given k, the MIMO channel coefficients $h_{i,\ell}$ are just the values of the different frequency responses at that specific subcarrier $H_{i,\ell;k}$, and our model holds true.

Keeping this in mind, all results (capacity, coding and decoding etc.) that we have discussed so far are to be intended *per (sub)carrier*. Channel estimation is also carried out per subcarrier with the aid of pilot symbols as in conventional OFDM. In addition, specific patterns of MIMO pilot symbols to perform individual estimation of $h_{i,\ell} = H_{i,\ell;k}$ are also described by wireless communication standards. The basic idea is that to estimate $h_{i,\ell}$, $\ell = 1, ..., N_R$ transmit antenna #i is activated with a specific pilot symbol known to the receiver, whilst all of the other antennas are switched off, so that no cross-interference terms appear at the receiving antennas. All wideband wireless digital communication technologies are *de facto* adopting multicarrier technologies, so that the narrowband modeling that we have adopted up to now is well justified, and does not constitute a limitation or an approximation by any means.

6.2.6 Ergodic Capacity and Outage on the Rayleigh Channel

In all of our computations regarding channel capacity we have assumed that the MIMO channel matrix **H** is known and constant throughout the whole communication session. As in conventional SISO links, we may have on the contrary a variable channel, either because we intend to capture the variability of the possible propagation scenarios and then evaluate an average capacity, or because there is a slow (wrt to the symbol or packet time) motion between TX and RX. In this section we try to address this issue and come up with relevant results.

The Shannon capacity of a MIMO link is given by (6.62) where **H** is known and given. If we wish to capture the variability of the MIMO channel, we have to assume on the contrary that **H** is a *random matrix*, i.e., a matrix whose entries are random variables, each one modeling the fading channel between two antennas. A reasonable model for each channel gain $h_{i,\ell}$ is that is is a complex independent-component Gaussian random variable, i.e., Rayleigh amplitude and uniform phase. A further simplifying assumption is that the entries of the matrix are statistically independent. While the Rayleigh model is very well verified in the practice, the latter assumption about independence of the channel gains is not completely justified since, as can be easily imagined, the actual geometry of the TX/RX antennas induces a non-negligible dependency between the fading process of two antennas - think of the correlation between the fading processes of two RX antennas that are rigidly connected to a single device. More sophisticated physical modeling is needed to capture this dependence, and is outside the scope of these notes. In, general the independency assumption leads to optimistic results, since a purely random matrix is with high probability higher-rank than a matrix with statistical dependence across entries.

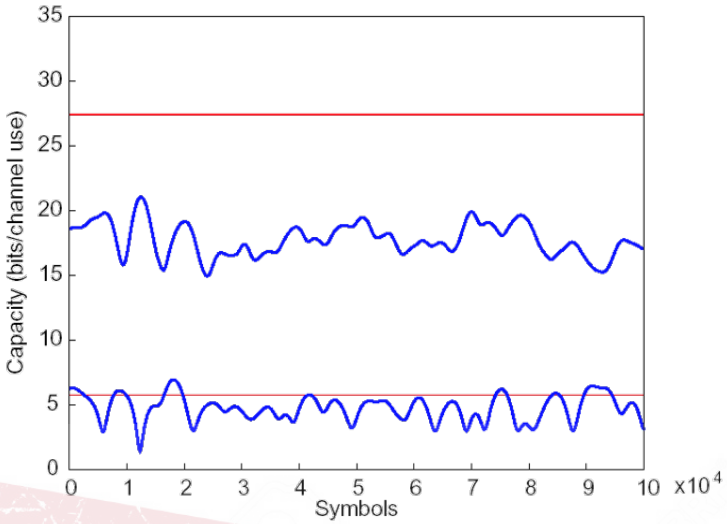


Figure 6.10 Instantaneous and Average (Ergodic) Capacity of a mobile MIMO Link

In any case, the *average* capacity of the MIMO link in the situation above (random channel with no TX CSI) needs averaging of the "instantaneous" capacity that assumes a given channel matrix:

$$c_E = \mathrm{E}_{\mathbf{H}} \left\{ \log_2 \det \left(\mathbf{I}_{N_T} + \frac{P_T}{N_T \sigma_w^2} \mathbf{H}^\dagger \mathbf{H} \right) \right\} \text{ bit/c.u. } , \quad N_T < N_R$$

$$c_E = \mathrm{E}_{\mathbf{H}} \left\{ \log_2 \det \left(\mathbf{I}_{N_R} + \frac{P_T}{N_T \sigma_w^2} \mathbf{H}\mathbf{H}^\dagger \right) \right\} \text{ bit/c.u. } , \quad N_T > N_R \tag{6.86}$$

This average capacity is also called *ergodic capacity*, because it is usually interpreted as the long-term average capacity experienced across a long communication session out of a MIMO link where channel variability is caused by time-variation of the channel due to relative motion. Explicit computation of this quantity is complicated also in the case of independent fading. Just to give the feeling of it, we report in Fig. 6.10 an example (by Schlegel) of a simulated long-term transmission on a MIMO channel. The figure display instantaneous capacity of a 2×2 (below) and a 8×8 link (blue curves) compared to the (overestimated) ergodic capacity (red line) of the same links with independent-entry fading matrix. The simulation is carried out with a simulator of the radio channel that models the dependence of the channel gains, and provide realistic results, in the following conditions: speed = 50 km/h, E_s/N_0=10 dB, f_0=2.4 GHz, distance between antenna elements= $5\lambda_0$=62.5 cm. The capacity is considerably time-variant, according to the specific channel status at a certain time, and may be far form the theoretical average (ergodic) value with independent fading.

Instead of computing the ergodic capacity and designing a link based on this pessimistic value, it may be preferable, as happens any time we deal with a long-term fading channel, to deal with the *outage probability*, i.e., the fraction of time during which the capacity of the link is below a certain (target) capacity threshold c_{th}:

$$P_{out} \triangleq \Pr\{c \leq c_{th}\} \tag{6.87}$$

This outage probability depends of course on the statistics of the channel matrix, and its exact formulation is again very complicated - it is usually evaluated assuming that c is a Gaussian random variable with average value equal to c_E, and with a variance that can be computed in close form - we will not deal with such topic here.

6.3 Beamforming, Space-Division Multiple Access, and Multiuser MIMO

We can also regard MIMO communications from a different perspective. Let us consider the uplink of a cellular network, assuming a receiving station with a relatively large number of antennas $N_R > N_T$, and let us imagine that instead of referring to a point-to-point link with a single (mobile) transmitting terminal equipped with N_T antennas, we change our scenario into one where we have N_T mobile terminals with a single antenna each - a kind of "distributed" MIMO. What we are in reality thinking of is clearly a kind of *multiple access* to service N_T conventional single-antenna users in a cell, all operating on the same bandwdith. Relation (6.53) tells us after all that capacity scales nicely and may be enough, since it is roughly proportional to the number of users N_T. We call this scenario Multiuser MIMO - our example focused on the uplink, but it can be easily reversed for the downlink with the same outcome.

What is fundamental in this view is that the resource that we exploit to carry out multiple access is not time, frequency, or code - it is just... *antennas*, so we need to know more about them, and we will also discover different perspectives on multiuser MIMO than the one we have just introduced.

6.3.1 Radiation pattern of an antenna

The possibility to serve multiple users with a MIMO-like setup is easily interpreted in the traditional context of *antenna beamforming*. As is known, the far-field of any radiating element can be characterized by a radiation pattern. Assume that our antenna is placed at the origin of a reference system as in Fig. 6.11, that it is radiating a total power P_T at the frequency $f_0 = c/\lambda_0$, and that this power is uniformly distributed in space. This means that at distance r from the antenna (where $r \gg \lambda_0$ so that we experience the far-field of the antenna) the power flux of the EM wave in free space is $P_T/(4\pi r^2)$, i.e., the total radiated (transmitted) power is uniformly distributed on a spherical surface of radius r. This is called an isotropical or omnidirectional antenna that sends out RF power along any direction with the same intensity, and represents a reference, theoretical device that is very hard (or unnecessary) to implement.

Example 6.50

Assume that we wish to build an antenna for a digital terrestrial TV broadcasting system. By definition, the antenna has to be omnidirectional in the X-Y plane to be able to address all of the subscribers, independent of their own location, or in other words the emission of the antenna has to be uniform wrt the *azimuth* angle ϕ (see Fig. 6.11). But, there is no need to send out power towards the sky or the ground, so that we had better designing an antenna whose emission is *not* uniform wrt the *elevation* angle θ. On the contrary, we should concentrate the power in a small angle with θ ranging form 0 (the ground level) to a few degrees to address the users at some height above the ground. If we figure out the distribution of the radiated power we imagine kind of a round "doughnut" lying on the ground, as in Fig. 6.12.

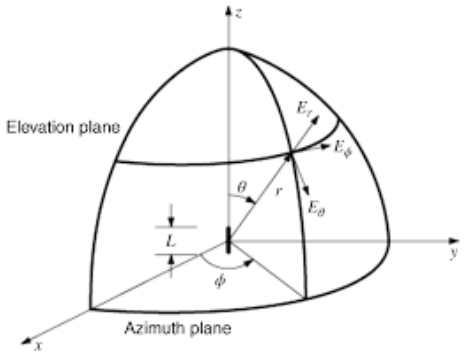


Figure 6.11 Antenna Reference system

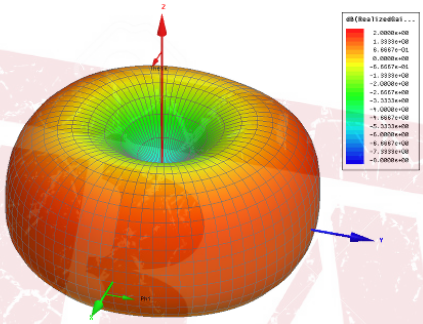


Figure 6.12 Irradiazione di un'antenna per radiodiffusione

What we have described in plain words is the so-called radiation pattern of an antenna, i.e., the distribution of radiated power across the azimuth and elevation angles. The pattern is not uniform in general, so as to have "privileged" directions of radiation, like in the broadcasting example above. The same applies to a satellite radiating towards the ground from a geostationary equatorial orbit at 35,786 km above the Earth surface, where it appears fixed in the sky from an observer on the ground (the TV dishes...). Considering that the Earth radius is roughly 6,378 km, the antenna on board the satellite has to concentrate its emission in a cone whose aperture angle is $2\sin^{-1}(6370/(35786+6378))$=17.3 degrees - otherwise some power is wasted (Fig. 6.13).

As is sketched in Fig. 6.12, the antenna pattern $G_T(\phi;\theta)$ is a function that describes how the radiated power is distributed across the different directions in space, defined by the angular (azimuth, elevation) coordinates. The pattern is normalized so that

$$\int_{\theta=-\pi/2}^{\pi/2}\int_{\phi=-\pi}^{\pi} G_T(\phi;\theta)d\phi d\theta = 1 \tag{6.88}$$

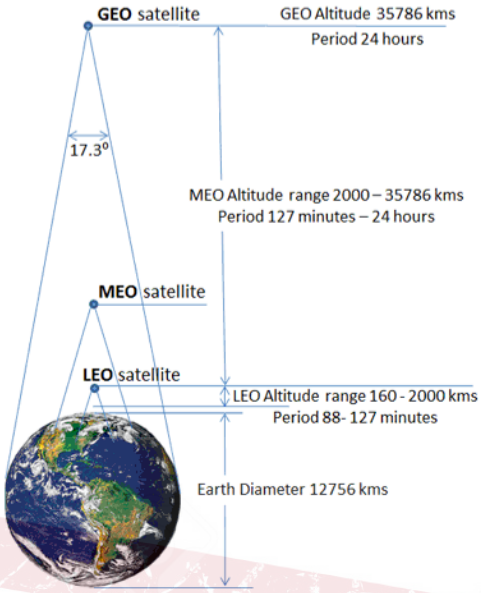


Figure 6.13 Geostationary satellite emission

With this condition, the power flux in a certain direction $(\phi;\theta)$ at distance r from the antenna is given by

$$p_T(\phi;\theta) = \frac{P_T}{4\pi r^2} \cdot G_T(\phi;\theta) \tag{6.89}$$

There will be certain directions $(\phi_0;\theta_0)$ for which $G_T(\phi_0;\theta_0) > 1$, so that the antenna has a *gain* wrt the isotropic radiating elements - this is why the radiation pattern is also called the *gain* of the antenna. Often, what is given of a certain antenna is just the maximum value of the gain $G_T \triangleq \max_{(\phi;\theta)} G_T(\phi;\theta)$, equally often expressed in dB, and the direction for which the maximum is attained is called the main antenna direciton.

The reasoning that we have just carried out for a transmitting antenna is also reciprocally valid for a receiving element. If the receiving antenna is ideally isotropic, and it is subject to a power flux p_T coming form *any* direction $(\phi;\theta)$, then the collected radio power is equal to $P_R = p_T \cdot A_e$ where A_e is the equivalent area of the antenna - a parameter that is related to the physical size of the element. For a conventional TV dish, the equivalent area is just a fraction close to 1/2 (that accounts for different factors like geometry of the parabolic antenna and the overall electric efficiency) of the area of the dish. But, the receiving antenna is not in the practice omnidirectional, so that the equivalent are does depend on the direction under which radio power is received: $A_e = A_e(\phi;\theta)$. By exploiting a reciprocity condition, the equivalent area is related to the antenna gain as follows:

$$A_e(\phi;\theta) = \frac{\lambda_0^2}{4\pi} \cdot G_R(\phi;\theta) \tag{6.90}$$

so that the well-known Friis' general equation for the received power on a radio link turns out to be

$$P_R(\phi;\theta) = \frac{P_T}{4\pi r^2} \cdot G_T(\phi;\theta)\frac{\lambda_0^2}{4\pi} \cdot G_R(\phi;\theta) = \frac{P_T}{(2\pi r/\lambda_0)^2} G_T(\phi;\theta) G_R(\phi;\theta) \tag{6.91}$$

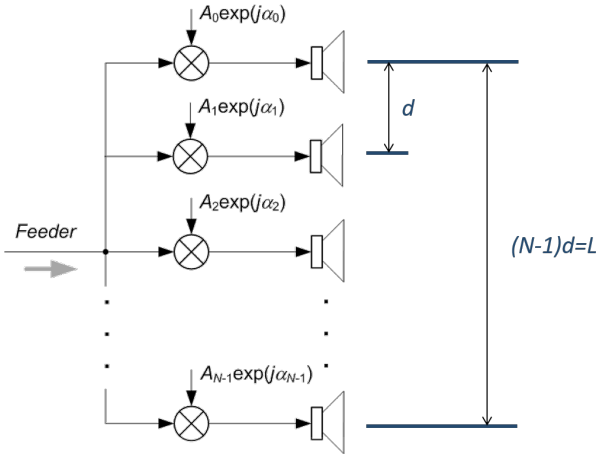


Figure 6.14 Linear Array Antenna

6.3.2 Linear Array Antenna and Array Factor

The examples above (especially the one with the geostationary satellite) tell us that we may need extremely focused or *directive* antennas for special needs, i.e., antennas with high gain along a certain prescribed direction and no (or very low) emission on the others. We may also wish to change at will the focus of the antenna so that the maximum-gain direction is steered in real time, possibly with no need of mechanically moving parts (what is called *electronic steering* of the antenna), just as happens with the roof-top mounted antenna for receiving satellite broadcasting by the car. How can we do this ?

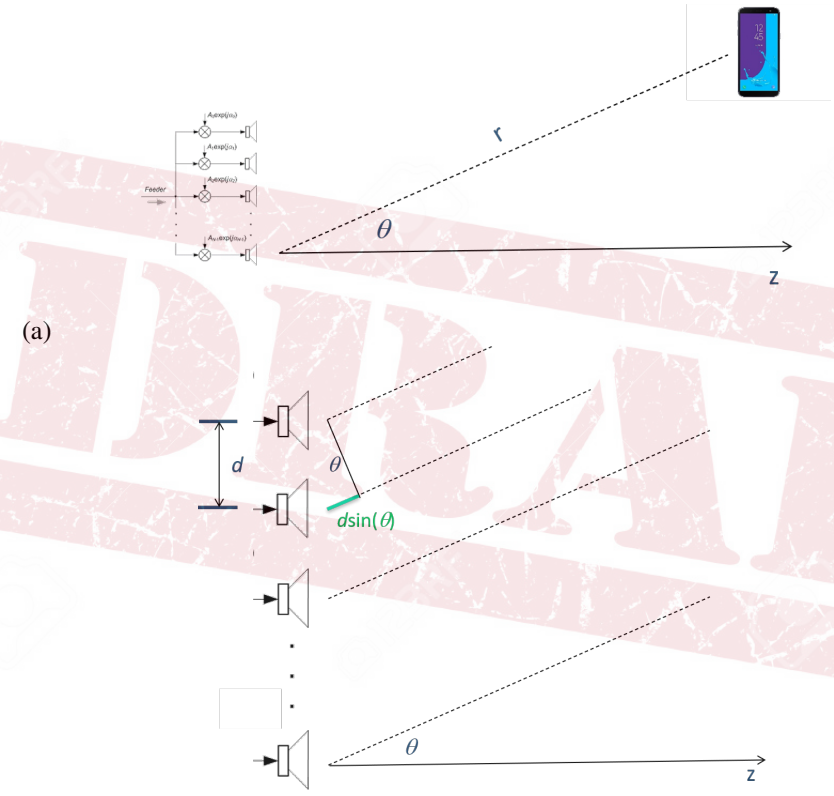
Figure 6.14 shows the description of a linear array antenna systems. It is made of N equal radiating elements (with the same gain $G(\phi;\theta)$) placed on a line at a regular distance d that is comparable with the wavelength λ_0. The signal that is fed to each element may also be changed in amplitude and/or phase wrt to the reference - for the moment, let us just assume that all phases are equal ($\alpha_i = 0\ \forall i$ for simplicity) and all amplitudes are equal ($A_i = 1\ \forall i$). Assuming a transmitting antenna, at a large distance $r \geq 2L^2/\lambda_0$ a receiving device is subject to the monochromatic progressive plane wave

$$E_{RF}(z;t) = E_M \cos\left(2\pi f_0 t - \frac{2\pi}{\lambda_0} z\right) \tag{6.92}$$

whose baseband equivalent is

$$E(z) = E_M \exp\left(-\jmath \frac{2\pi}{\lambda_0} z\right) \tag{6.93}$$

We make now an example wherein the array elements are placed vertically on the Y-axis, as is shown in Fig. 6.15 (a), and we take into consideration the elevation angle θ (given the antenna layout, there is no dependence of the array emission on ϕ). The received field $E_A(z)$ is the composition of the N fields $E_i(z)$, $i = 0, ..., N-1$ separately emitted by the N radiating elements. We also see from Fig. 6.15 (b) that the path traveled by each wave emitted by the different elements is different: the $i+1$ path is longer then the "previous" $i-th$ path by the displacement $d\sin(\theta)$. This difference is very small wrt to r, so in terms of amplitude of the received signal, this displacement can be neglected, and we can say



(a)

(b)

Figure 6.15 Linear Array Antenna Functioning

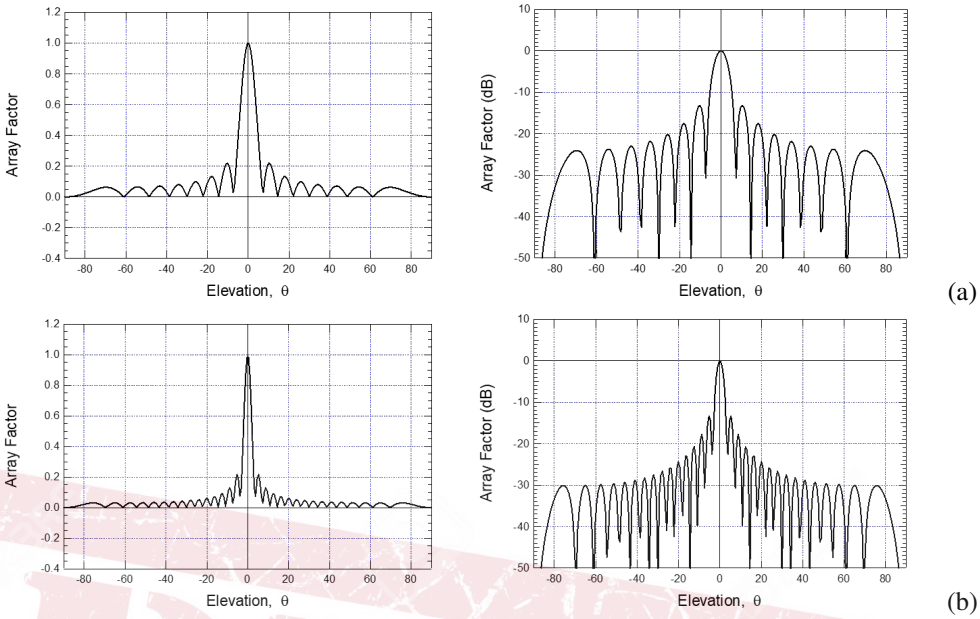


Figure 6.16 Array Factor Examples - $d = \lambda_0/2$, $N = 16$ (a), and $N = 32$, (b)

that all emissions are received with the same strength. On the contrary, the displacement is comparable to the wavelength (since d is such), so what we cannot neglect is the phase shift $2\pi d \sin(\theta)/\lambda_0$ between "adjacent" components, caused by the path difference. All in all, the total received field $E_A(z)$ is

$$E_A(z) = \sum_{i=0}^{N-1} E_0(z) \exp\left(-\jmath 2\pi i \frac{d\sin(\theta)}{\lambda_0}\right) = E_0(z) \frac{1 - \exp\left(-\jmath 2\pi N \frac{d\sin(\theta)}{\lambda_0}\right)}{1 - \exp\left(-\jmath 2\pi \frac{d\sin(\theta)}{\lambda_0}\right)}$$

$$= E_0(z) \exp\left(-\jmath \pi N \frac{d\sin(\theta)}{\lambda_0}\right) \frac{\sin\left(\pi N \frac{d\sin(\theta)}{\lambda_0}\right)}{\sin\left(\pi \frac{d\sin(\theta)}{\lambda_0}\right)} \tag{6.94}$$

where $E_0(z)$ is the received field from element #0. The received power from the antenna array is proportional to $|E_A(z)|^2$, so we see from (6.94) that the array is introducing a directive factor on the total antenna emission as follows:

$$A_F(\theta) \triangleq \left| \frac{\sin\left(\pi N \frac{d\sin(\theta)}{\lambda_0}\right)}{N \sin\left(\pi \frac{d\sin(\theta)}{\lambda_0}\right)} \right|^2 \tag{6.95}$$

where we have introduced the normalization factor N at the denominator so as $A_F(0) = 1$. This quantity is called *array factor*, and its effect is focusing the antenna emission along the direction $\theta = 0$. Figure 6.16 shows the array factor for $N = 16$ and $N = 32$ where the directive effect of the array is apparent, and where we also see that such effect increases for increasing N. We also depict in Fig. 6.17 how the array factor looks in space - the directivity in elevation is apparent, as it is also the omnidirectionality in azimuth.

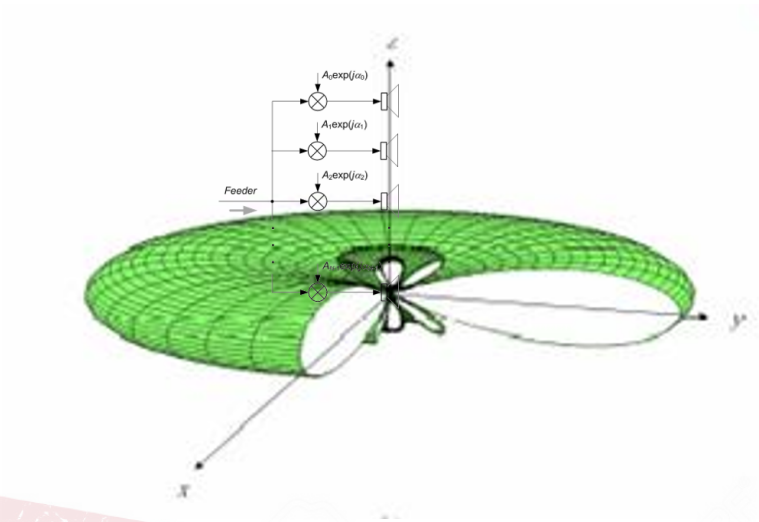


Figure 6.17 3D Array Factor Shape - $d = \lambda_0/4$, $N = 10$

Until now, we only have dealt with the array factor A_F of the linear array showing that it contributes to give strong directionality to the antenna. In reality, the actual antenna pattern $G_A(\phi, \theta)$ of the linear array is made of the array factor *times* the native antenna gain of the single radiating element:

$$G_A(\phi, \theta) = G_T(\phi, \theta) \cdot A_F(\theta)$$

As a simple example, we can take a linear array of $\lambda/2$ dipoles, i.e., simple radiating elements made of a wire whose length is half the wavelength as in Fig. 6.18. The $\lambda/2$ dipole has a pattern given by

$$G_T(\phi; \theta) = \frac{\cos^2\left(\frac{\pi}{2}\sin(\theta)\right)}{\cos^2(\theta)} \tag{6.96}$$

The pattern is omnidirectional in the X-Y plane, and has a doughnut shape as shown in Fig. 6.18 (a). The array has a total gain

$$G_A(\phi; \theta) = \left| \frac{\sin\left(\pi N \frac{d\sin(\theta)}{\lambda_0}\right)}{N \sin\left(\pi \frac{d\sin(\theta)}{\lambda_0}\right)} \right|^2 \frac{\cos^2\left(\frac{\pi}{2}\sin(\theta)\right)}{\cos^2(\theta)} \tag{6.97}$$

represented in Fig. 6.18 (b). The directionality enhancement of the antenna is apparent: we can say that with the array we have synthesized a narrow radio beam to selectively (i.e., directionally) address only receivers on the ground (at 0 elevation). In addition to the main beam, the antenna also bears "side lobes" in its diagram that represent undesired secondary beams. As a rule, the larger is N, the lower are the side beams.

We will not insist further on the study and design criteria of the different types of linear arrays, nor will we introduce more complicated 2D arrays to create directive arrays both along the *vertical* and the *horizontal* plane, with a *pencil-shaped beam*. We will just go on in our study observing that the linear array as we have described it enhances directionality of

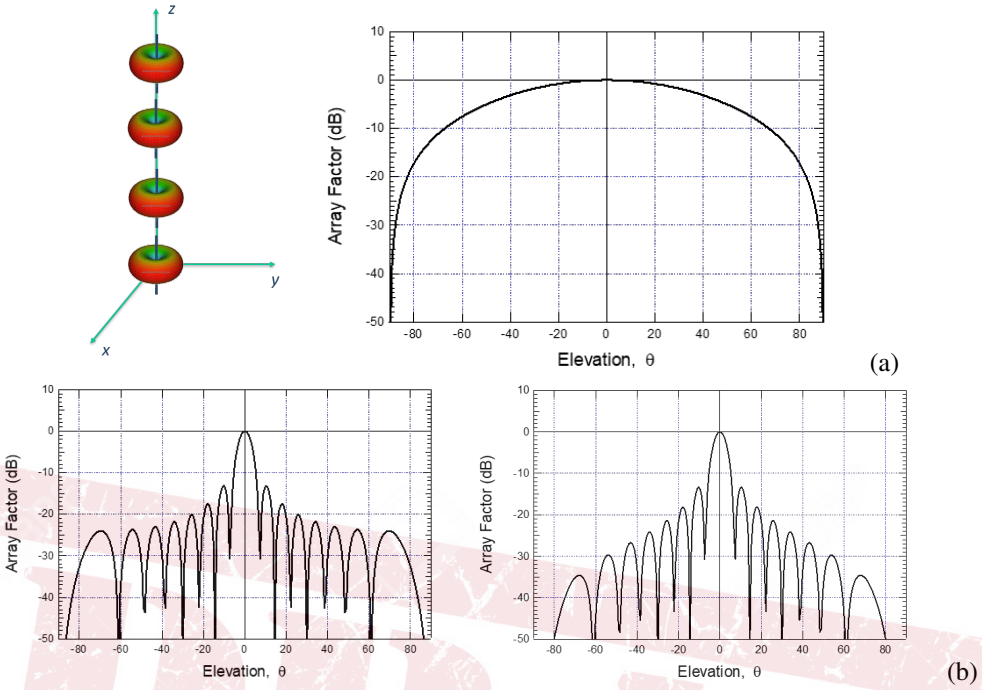


Figure 6.18 Linear Array of Dipoles with N=16 and $d = \lambda_0/2$ - Setup and individual antenna pattern (a); Array Factor and Total Antenna Pattern (b).

the antenna along the direction θ=0. Can we address an airplane with the beam synthesized by the array that is seen from the antenna at a certain elevation angle $\theta_0 \neq 0$ on ground, for instance θ_0= 15 degrees, without the need to physically tilt the antenna? In other words, can we synthesize or *form* a beam with an arbitrary elevation θ_0 ? This is the subject of the next section.

6.3.3 Beamforming and Steering Vector

The solution to the *beamforming* problem that we have just stated lies in the possibility to feed the different elements of the array with signals undergoing appropriate phase and/or amplitude shifts. This technology has been used for tens of years in the field or radar, radioastronomy, and satellite communications under the name of *phased arrays*. More, recently they have also been used in cellular radio base stations with the name of *smart antennas*.

Let us first take the simplified approach of the phased arrays, wherein $A_i = 1 \; \forall i$, and let us try to steer the main beam of the antenna away from θ=0 towards $\theta = \theta_0$ by properly adjusting the phase shifts α_i. Including the phase shifts in the computation of the array factor, we get

$$A_F(\theta) = \left| \sum_{i=0}^{N-1} \exp\left[-\jmath \left(2\pi i \frac{d \sin(\theta)}{\lambda_0} - \alpha_i \right) \right] \right|^2 \tag{6.98}$$

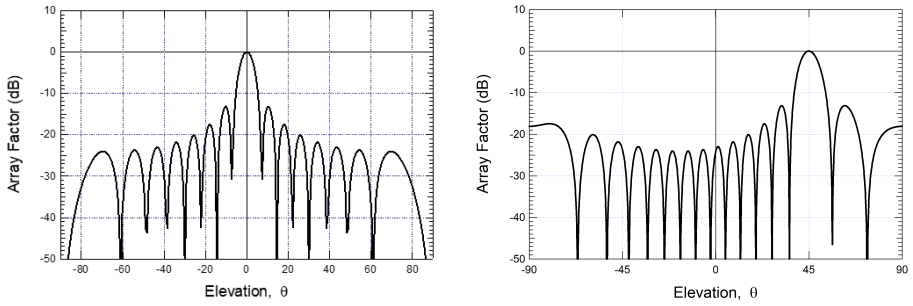


Figure 6.19 Example of un-Steered (a) and Steered (b) Beams - $d = \lambda_0/2$, $N = 16$, $\theta_0 = \pi/4$

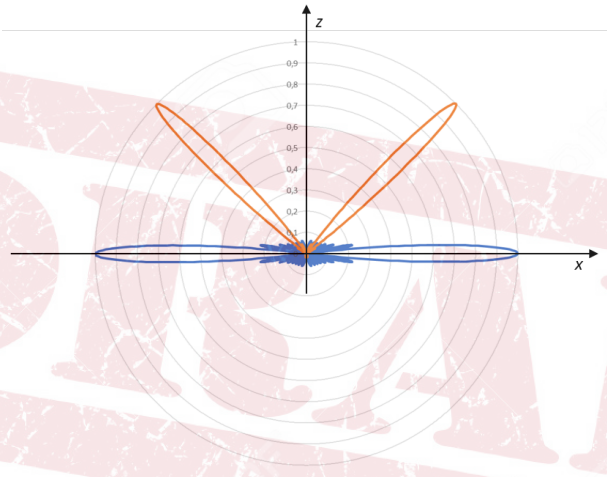


Figure 6.20 Comparison of Steered and un-Steered Beams on a polar chart

Steering the main beam (i.e., the main maximum of the array factor) from 0 to θ_0 can be trivially done by letting

$$\alpha_i = 2\pi i \frac{d \sin(\theta_0)}{\lambda_0} \tag{6.99}$$

This is not a simple (cyclical) shift of the previous array factor wrt θ as we would expect, but it works - the result is

$$A_F(\theta) \triangleq \left| \frac{\sin\left(\pi N \frac{d(\sin(\theta)-\sin(\theta_0))}{\lambda_0}\right)}{N \sin\left(\pi \frac{d(\sin(\theta)-\sin(\theta_0))}{\lambda_0}\right)} \right|^2 \tag{6.100}$$

Figure 6.19 shows the array factor for the same array as in Fig 6.16 with N=16, but now with $\theta_0 = \pi/4$, i.e., 45 degrees. The two array factors (steered and un-steered) are compared in the polar diagram (also called "radar" chart) shown in Fig. 6.20 that is more commonly used to display antenna patterns. By changing in real time the values of the phase shifts α_i, the beam can be made adaptive and variable in time, for instance to track instant by instant the position of a satellite in the sky from the antenna mounted on top of a car, and

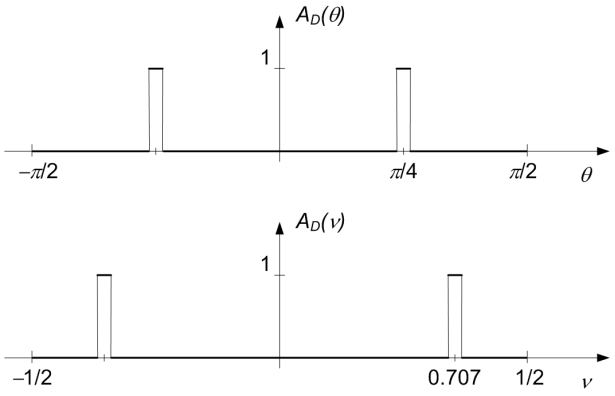


Figure 6.21 Ideal Array Factor in the θ and ν domains

achieve optimum reception of a radio broadcasting program - this is called an *electronically-steerable* antenna. Applying variable and controlled phase shift is relatively easy: it can be done either changing the phase of a local oscillator or at baseband via digital signal processing.

The design criteria for phased arrays in terms of number N and spacing d of the radiating elements, as well as the phase values α_i, are well studied and consolidated, and we will not insist further on this topic here. A typical value for the spacing d is $d = \lambda_0/2$ (as we have already adopted in many examples) because it leads to some simplification in the design criteria and leads to relatively small but efficient arrays. In this respect, and especially if the steering system is operated at baseband (so that there is total freedom of selecting appropriate phase shifts α_i and amplitudes A_i), it makes sense to state the general beamforming problem:

"Given a desired, ideal beam pattern $A_D(\theta)$ like for instance in Fig. 6.21 in terms of beam angle θ_0 and aperture $\Delta\theta$, find the set of coefficients $g_i \stackrel{\triangle}{=} A_i \exp(\jmath\alpha_i)$ such that the actual array factor

$$A_F(\theta) = \left| \sum_{i=0}^{N-1} g_i \exp\left[-\jmath 2\pi i \frac{d \sin(\theta)}{\lambda_0} \right] \right|^2 \tag{6.101}$$

is the closest that is possible, under some metrics, to the desired one."

The parameter θ_0 dictates antenna steering, whilst $\Delta\theta$ (the beam *aperture*) regulates its selectivity.

If $d = \lambda_0/2$,

$$A_F(\theta) = \left| \sum_{i=0}^{N-1} g_i \exp\left[-\jmath 2\pi i \frac{\sin(\theta)}{2} \right] \right|^2 = \left| \sum_{i=0}^{N-1} g_i \exp\left[-\jmath 2\pi i \nu \right] \right|^2 \tag{6.102}$$

where we let $\nu \stackrel{\triangle}{=} \sin(\theta)/2$. Equation (6.102) exhibits a strict analogy to the computation of the frequency response of an FIR digital filter whose impulse response is g_i, and the analogy is reinforced by the fact that while θ spans $[-\pi/2, \pi/2)$, ν spans $[-1/2, 1/2)$ just like the normalized frequency f/f_s (f_s is the sampling frequency) in the spectrum of digital signals.

This fundamental remark is the key to a number of design methods of the so called *steering vector* $\mathbf{g} \triangleq [g_0\ g_1\ ...\ g_{N-1}]$ (i.e., the ordered set of complex-valued array coefficients) that implements (an approximation of) the desired array factor. The beamforming problem is reduced to an already well-known FIR design issue, with all the relevant tips and tricks that the reader is supposed to know. Changing the steering vector means changing the beam pattern of the antenna. Also, just like we can design multi-band (also called *comb* FIR filters), we can also have multi-beam antennas to address more than one receiver at a time at different elevations. Why not trying to generalize this view and conceive an array system with very many steerable beams? This is what in wireless communications is called *smart antenna*, and is the basis for the so called Space Division Multiple Access (SDMA) technology, very much related to the main topic of this Chapter, i.e., MIMO communications.

6.3.4 Space-Division Multiple Access (SDMA) and Multiuser MIMO

To better understand SDMA, let us first rotate our array so that it lies on the X-axis. What we get is an antenna that is omnidirectional in the vertical plane (elevation), but highly directional on the X-Y plane (azimuth), so that now (assuming no steering)

$$A_F(\phi) \triangleq \left| \frac{\sin\left(\pi N \frac{d\sin(\phi)}{\lambda_0}\right)}{N\sin\left(\pi \frac{d\sin(\phi)}{\lambda_0}\right)} \right|^2 \tag{6.103}$$

If we wish to represent A_F we end up with the same chart as in Fig. 6.20, where the plane on which we represent the array factor is now the *horizontal* X-Y plane - just like projecting the beam on a map. Imagine now that the origin of the axes is the location of a radio base station (RBS) of a cellular network equipped with our array antenna, and that the RBS is serving a population of U (mobile) user terminals scattered on the surface of the cell, as sketched in Fig. 6.22. Each terminal is seen by the RBS along a certain direction ϕ_u, $u = 1, ..., U$, and is transmitting with a conventional omnidirectional antenna a constellation symbol $s_m^{(u)}$ at time m as usual.

Assuming a frequency-flat channel (or if you wish OFDM transmission), we can recover the symbol of user # u without interference from the other users if we apply a steering vector $\mathbf{g}^{(u)T} = [g_0^{(u)}\ g_1^{(u)}\ ...\ g_{N-1}^{(u)}]$ to the received signal impinging on the array so that we synthesize a beam along the very direction of that user: $g_i^{(u)} = \exp(\jmath 2\pi i d \sin(\phi_u)/\lambda_0)$. The different users can be separated leveraging on the diversity of their own (2D) *space* location - that's the rationale of the name SDMA - eve if they "occupy" the same frequency band and/or time slot. Of, course to apply the proper steering vector, the RBS needs to know the location of the user, just like in conventional TDMA (FDMA) the RBS has to know the time slot (frequency channel)a particular user has been allocated.

How can we relate this naive geometric interpretation of SDMA with the rigorous approach of MIMO systems that we already know? After all, we are anyway speaking of exploiting many antenna elements in both cases. If we call as usual $\mathbf{r}$ the output of the RX array elements in the RBS, taking back the modeling in (6.94), we can say that in the uplink of the cell we have U $1 \times N_R$ MIMO links (also called SIMO, single-input multiple-output), so that

$$\mathbf{r} = \sum_{\ell=1}^{U} s_m^{(\ell)} \cdot \mathbf{H}^{(\ell)} + \mathbf{w} = \sum_{\ell=1}^{U} s_m^{(\ell)} \cdot \mathbf{h}^{(\ell)} + \mathbf{w} = s_m^{(u)}\mathbf{h}^{(u)} + \sum_{\ell \neq u} s_m^{(\ell)} \cdot \mathbf{h}^{(\ell)} + \mathbf{w} \tag{6.104}$$

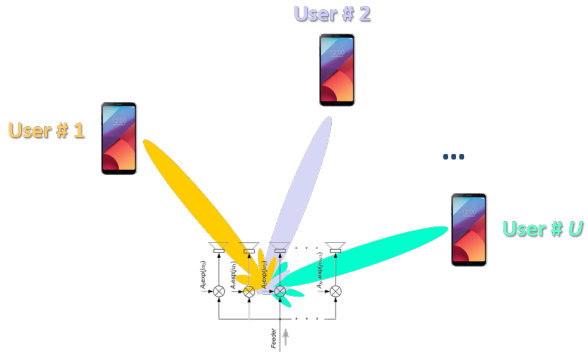


Figure 6.22 Principle of operation of SDMA

and where the $N_R \times 1$ channel coefficient matrix for user # ℓ is actually the vector

$$\mathbf{h}^{(\ell)} = \begin{bmatrix} h_1^{(\ell)} \\ \vdots \\ h_{N_R}^{(\ell)} \end{bmatrix} \qquad h_i^{(\ell)} = \bar{h}^{(\ell)} \exp(-j2\pi id \sin(\phi_\ell)/\lambda_0) \quad , \quad i = 1, \ldots, N_R$$

with $\bar{h}^{(\ell)}$ è the channel coefficient we would have for the conventional 1×1 link.

Equation (6.104) shows that in general there may be Multiple Access Interference (MAI) between users that are (all) using the same band at the same time. Let us not consider this MAI; since we know that the desired user signal comes from direction ϕ_u, we do beamforming, and we apply a steering vector $\mathbf{g}^{(u)}$ according to (6.99)

$$\mathbf{g}^{(u)} = \text{vect}\left\{\exp(j2\pi id \sin(\phi_u)/\lambda_0)\right\}_{i=1}^{N_R} \tag{6.105}$$

The output of the user #u beamformer, i.e., the relevant soft decision variable, is

$$z_m^{(u)} = \mathbf{g}^{(u)T}\mathbf{r} = \sum_{i=1}^{N_R} r_i \exp(j2\pi id \sin(\phi_u)/\lambda_0) \tag{6.106}$$

How can we compare this beamforming approach with MIMO detection? Our uplink BTS receiver should actually be a $N_R \times 1$ *diversity receiver* adopting an MRC optimal strategy with combining coefficients (6.10) $a_i^{(u)} = h_i^{*(u)}$:

$$\zeta_m^{(u)} = \sum_{i=1}^{N_R} r_i h_i^{*(u)} = \bar{h}^{*(u)} \sum_{i=1}^{N_R} r_i \exp(j2\pi id \sin(\phi_u)/\lambda_0) \tag{6.107}$$

This combination, apart form the coefficient $\bar{h}^{*(u)}$ is *equivalent to beamforming*. Application of the beamforming steering vector is equally optimum wrt to noise (provided it is corrected with the channel coefficient $\bar{h}^{*(u)}$). And, it is also optimum wrt to MAI, because of the directionality property of the synthesized antenna that cancels the other users' signals coming from different directions - same as with FDMA, when we use a *bandpass* matched filter that is optimum wrt noise for user # u, and also isolates spectrum # u from those of the other users.

What we have illustrated is just the principle of SDMA: users that exploit the same frequency band at the same time are actually separable because they occupy different locations in space. Separation of users can be done provided that space diversity is exploited, i.e., the signals emitted by the different user terminals are received by a MIMO (array) antenna whose elements are in their turn separated in space and can exploit space diversity. Of course the approach is a bit naive: users can be actually separated with no (or negligible) interference only iff their angular distance $\Delta\phi$ is larger than the beamwidth (aperture) that the receiving RBS antenna is capable to synthesize (the depends on the array size N_R). Generally speaking, to be able to fully exploit space diversity, the number of receiving antennas at the RBS N_R has to be much larger than the number of users U. This condition is clear (also) from the point of view of beamforming or that of smart antennas: as we can see in Fig. 6.22, the smart antenna for SDMA must be able to synthesize as many narrow-aperture beams as the number of users that are to be served in a cell to keep them orthogonal, and this can only be effectively achieved if $N_R \gg U$.

Is there any difference between SDMA and MU-MIMO? SDMA is based just on geometric consideration, trying to separate users on the basis of location information as above - it is completely blind about users' nature and requirements and/or the status of the users' channels, and boils down to (adaptive) beamforming as above. With MU-MIMO, we can try to design the steering vector following a criterion a little bit more refined than just location-based beamforming. Let us combine (6.104) with (6.106) to get the expression of the soft output $z_m^{(u)}$ of the BTS receiver for user (u) at time m:

$$z_m^{(u)} = \mathbf{g}^{(u)T}\mathbf{r} = s_m^{(\ell)}\mathbf{g}^{(u)T}\mathbf{h}^{(u)} + \sum_{\ell \neq u} s_m^{(\ell)} \cdot \mathbf{g}^{(u)T}\mathbf{h}^{(\ell)} + W_m^{(u)} \tag{6.108}$$

where $\mathbf{g}^{(u)T}$ is the generic steering vector for user # u and where we have clearly separated the contributions of i) useful signal, ii) multiple-access interference, iii) noise. We can assume that the interference term, coming form a number of independent contributions, follows Gaussian statistic by virtue of the central limit theorem, so that we are basically faced with an AWGN channel for which the computation of the Shannon capacity is relatively simple:

$$c^{(u)} = \log\left(1 + \text{SINR}_u\right) = \log\left(1 + \frac{S_2|\mathbf{g}^{(u)T}\mathbf{h}^{(u)}|^2}{2\sigma_W^{2,(u)} + S_2\sum_{\ell \neq u}|\mathbf{g}^{(u)T}\mathbf{h}^{(\ell)}|^2}\right) \tag{6.109}$$

where $\sigma_W^{2,(u)} = \|\mathbf{g}^{(u)}\|^2\sigma_w^2$. The steering problem can now be (re)formulated as follows: find that set of (precoding/steering) vectors that maximizes some aggregate capacity in the cell. We say "aggregate" because in general selection of a particular value of $\mathbf{g}^{(u)}$ also affects the capacity of users $\neq u$ because of MAI, so that *individual* capacity maximization of a single user does not make any sense.

The usual target function to be maximized is the weighted total capacity (*weighted sum rate*)

$$c_{wsr} \triangleq \sum_{u=0}^{U-1} \alpha_u c^{(u)} \tag{6.110}$$

where $0 \leq \alpha_u \leq 1$, is a priority coefficient for user u, $\sum \alpha_u = 1$, under a power constraint

$$\sum_{u=0}^{U-1} \|\mathbf{h}^{(u)}\| \leq H \tag{6.111}$$

The normalized steering vector resulting from this constrained optimization is

$$\mathbf{g}_{wsr}^{(u)} = \frac{(\mathbf{I} + \sum_{\ell \neq u} q_\ell \mathbf{h}_\ell \mathbf{h}_\ell^\dagger)^{-1} \mathbf{h}_u}{\|(\mathbf{I} + \sum_{\ell \neq u} q_\ell \mathbf{h}_\ell \mathbf{h}_\ell^\dagger)^{-1} \mathbf{h}_u\|} \tag{6.112}$$

where the coefficients q_u depend on the priorities α_u.

Notice that equation (6.108), that we have derived for the UL of our SIMO setup, applies with (almost) no modifications to the case of the DL as well. The vectors $\mathbf{g}^{(u)T}$ are now applied by the BTS to the constellation symbols $s_m^{(u)}$ *before* sending them out, but the individual decision variables $z_m^{(u)}$ bear exactly the same form - the UL and DL channels are reciprocal. In the context of the DL, applying the steering vectors is called *precoding*.

In addition to the receivers that we have just examined, which are based on a linear processing (steering vector) approach, more complicated yet effective receivers can be also designed. We think in particular, of *successive cancelation* receivers that, after performing a linear processing similar to what we have just seen, identify the user received with the maximum SNR, let's call it $\bar{u}$, detect the symbol $s_m^{(\bar{u})}$ by conventional hard-decision based on $z_m^{(\bar{u})}$, and then *reconstruct* the contribution of MAI $\hat{s}_m^{(\bar{u})}\mathbf{h}^{(\bar{u})}$ by $\bar{u}$ into (6.108). This contribution is subtracted (*canceled*) from the linear detector outputs of all of other users $\ell \neq \bar{u}$, after it is weighted with the relative steering vector $\mathbf{g}^{(\ell)T}$. Once this is done, the user in the remaining $U-1$ set whose SNR is the largest is identified, its data symbol is detected, its MAI contribution is reconstructed and canceled from the other $U-2$ and so on... The analysis of this approach, that reveals in certain situations very effective, is complicated and will not be considered here.

We will not even deal with the case wherein the various MU-MIMO links are in general $N_R \times N_T$ with $N_T > 1$. We will mention this possibility in the next section relating to a technology that is being developed worldwide, namely, the *Massive* version of MIMO.

6.4 The New Frontier: Massive MIMO

Nothing prevents us form envisaging a scenario where the mobile terminals, too, have a number of antennas $N_T > 1$. In this case there may be also some form of spatial multiplexing in the uplink, in which case we have to make sure that $N_R \gg U \cdot N_T$, and we should re-compute the Shannon capacity in these conditions - very complicated. On the other hand, the MU-MIMO constraint $N_R \gg U \cdot N_T$ leads naturally to the consideration of antenna systems with a very, *very* large number of elements - something that appears like science fiction, but that in the contrary is about to be implemented from 5G onwards under the name of Massive MIMO (MMIMO) technology.

Let us take back into consideration the ergodic capacity c_E on the Rayleigh channel (6.86). We know that its computation is complicated; what on the contrary is easy to evaluate, and is very informative, is the value of c_E when the number of antennas on one side of the MIMO link is very large. When either N_R or N_T gets very large, it is easy to understand that by the law of large numbers the products $\mathbf{H}^\dagger\mathbf{H}$ and $\mathbf{H}\mathbf{H}^\dagger$ in (6.86) tend to their own expected values $\mathrm{E}\left\{\mathbf{H}^\dagger\mathbf{H}\right\} = N_R\mathbf{I}_{N_T}$ and $\mathrm{E}\left\{\mathbf{H}\mathbf{H}^\dagger\right\} = N_T\mathbf{I}_{N_R}$, respectively, (since we assume here uncorrelated unit-power fading). To understand why, let us take the downlink-like case of $N_T \gg N_R$. Then,

$$c_E = \mathrm{E}_{\mathbf{H}}\left\{\log_2 \det\left(\mathbf{I}_{N_R} + \frac{P_T}{N_T\sigma_w^2}\mathbf{H}\mathbf{H}^\dagger\right)\right\}$$

On the other hand, the i,k element of $\mathbf{HH}^\dagger$ $(i,k=1,\ldots,N_R)$ is

$$\left(\mathbf{HH}^\dagger\right)_{i,k} = N_T \cdot \frac{1}{N_T}\sum_{\ell=1}^{N_T} h_{i,\ell}h_{k,\ell}^* \quad \rightarrow \quad N_T \mathrm{E}\{h_{i,\ell}h_{k,\ell}^*\} = N_T\delta[i-k] \tag{6.113}$$

therefore

$$c_E \rightarrow \log_2 \det\left(\left(1+\frac{P_T}{N_T\sigma_w^2}N_T\right)\mathbf{I}_{N_R}\right) = N_R \log_2\left(1+\frac{P_T}{\sigma_w^2}\right) \text{ bit/c.u.} \tag{6.114}$$

This expression tells us that having so many transmit antennas gives us the certainty to be able to attain the maximum multiplexing gain, equal in this case to N_R, and cancels the effect of Rayleigh fading: in the special case N_R=1, we see that the ergodic capacity of the link is in practice that of an equivalent AWGN channel with a signal power equal to P_T (see also (6.16)). This comes at the expense of the diversity gain, that is sacrificed on the altar of multiplexing.

Does this result bear any practical relevance? Yes indeed - a large number of transmitting antennas can be found in the downlink of a modern cellular network since the transmit antenna resides in the base station. In addition, the asymptotic value (6.114) does not actually need such a large value of antennas to be attained - $N_T \approx 10$ might be enough.

What if we consider the uplink of the same cellular network? In this case, it is N_R that tends to infinity, and

$$c_E \rightarrow \log_2 \det\left(\left(1+\frac{P_T}{N_T\sigma_w^2}N_R\right)\mathbf{I}_{N_T}\right) = N_T \log_2\left(1+\frac{N_R}{N_T}\frac{P_T}{\sigma_w^2}\right) \text{ bit/c.u.} \tag{6.115}$$

We see that we have the maximum multiplexing gain, now equal to N_T, but we also have a considerable diversity gain coming from the very many receive antennas that we have available. The diversity factor is in fact equal to N_R/N_T, i.e, the number of available receive antennas per transmit antenna, and adds up to the multiplexing gain to make capacity increase.

MMIMO becomes practical for millimeter-wave bands (f_0 higher that 30 GHz or λ_0 smaller than 1 cm) wherein the wavelength is so small that the radiating elements of the array can be very small as well (orders of mm), and packing tens or even hundreds of them in a compact and integrated component is viable - it is one of the main PHY layer technologies envisaged for 5G cellular networks.

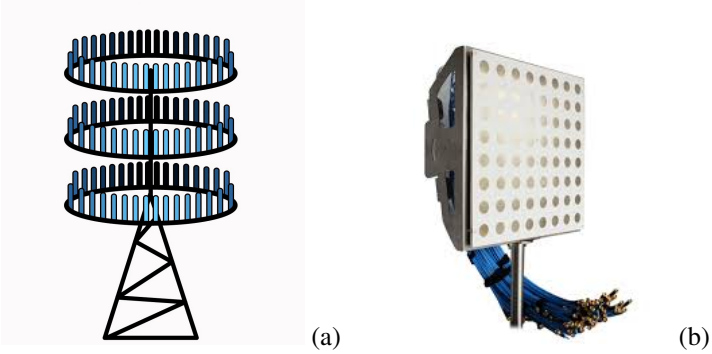
MMIMO lends naturally itself for a multiuser implementation. Think of a population of U users in a cell, each with a MIMO user terminal with N_T antennas, as an equivalent *fictitious* single user with of $U \cdot N_T$ antennas and total transmitted power UP_T, where P_T is the average transmitted power *per user*. We can re-use the ergodic asymptotic uplink capacity formula (6.115) to find out the total uplink multiuser MMIMO capacity in the cell:

$$c_{MM} = N_T \cdot U \log_2\left(1+\frac{N_R}{N_T\cdot U}\frac{U\cdot P_T}{\sigma_w^2}\right) = N_T\cdot U\log_2\left(1+\frac{N_R}{N_T}\frac{P_T}{\sigma_w^2}\right) \text{ bit/c.u.} \tag{6.116}$$

so that the average ergodic capacity *per user* is

$$c_{MM} = N_T \log_2\left(1+\frac{N_R}{N_T}\frac{P_T}{\sigma_w^2}\right) \text{ bit/c.u.} \tag{6.117}$$

exactly as in the single-user case, showing maximum multiplexing gain, diversity gain, Rayleigh fading cancelation, and finally complete decorrelation (orthogonalization) of the



(a) (b)

Figure 6.23 16×16 2D Array (a) and circulare array (b) as examples of MMIMO antennas

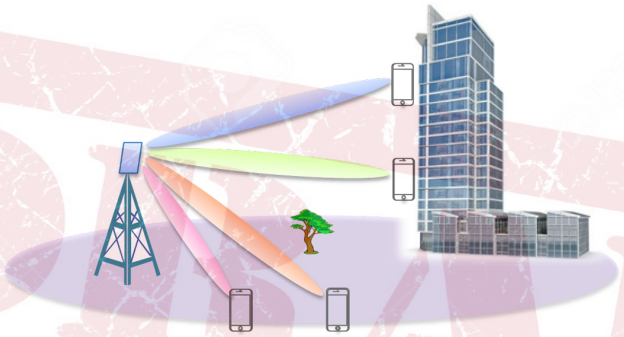


Figure 6.24 3D SDMA with MMIMO

potentially interfering users. Of course, the asymptotic regime is harder to attain since the number of users U also plays a role in the asymptotic condition $N_R \gg UN_T$.

The actual capacity that can be obtained with MMIMO is somewhat smaller than predicted by (6.117). This is not because the asymptotic regime is not reached; rather, it has to do with the fact that the modeling of the channel matrix with independent entries is not completely justified - the fading processes on the different antennas are correlated to some extent, so that $\mathbf{H}\mathbf{H}^\dagger \nrightarrow N_T\mathbf{I}$, and the capacity is reduced (since the users are not perfectly decorrelated).

A MMIMO antenna can be a 2D linear array, with as many as 16×16 radiating elements place on a regular grid, or a (set of) circular array(s) as in Fig. 6.23. With a 2D array, directionality is implemented not only on the horizontal plane as we have see with SDMA, but on the *vertical* plane as well, to cope for instance with high user density in skyscrapers of a densely populated urban area (Fig. 6.24).

APPENDIX G

WATER FILLING SOLUTION

We recall the problem to be solved in the maximization of MIMO Shannon capacity for spatial multiplexing with power allocation.

We are to find the set of allocated powers $P_1, P_2, ..., P_{N_T}$ such that

$$\begin{cases} c(P_1, ..., P_{N_T}) = \sum_{i=1}^{N_T} \log_2 \left(1 + \frac{d_i^2 P_i}{\sigma_w^2}\right) = \max \\ P_1 + P_2 + ..., P_{N_T} - P_T = 0 \end{cases} \tag{G.1}$$

To solve this problem, we introduce a Lagrange multiplier λ, and we build the following Lagrangian function:

$$\sum_{i=1}^{N_T} \log_2 \left(1 + \frac{d_i^2 P_i}{\sigma_w^2}\right) + \lambda \left(P_1 + P_2 + ..., P_{N_T} - P_T\right) \tag{G.2}$$

that we differentiate wrt to P_i $i = 1, \ldots, N_T$, and to λ, setting the derivatives equal to 0:

$$\frac{d_i^2/\sigma_w^2}{1 + \frac{d_i^2 P_i}{\sigma_w^2}} + \lambda = 0 \quad i = 1, ..., N_T$$

$$P_1 + P_2 + ..., P_{N_T} - P_T = 0 \tag{G.3}$$

Elaborating,

$$\frac{1}{\sigma_w^2/d_i^2 + P_i} + \lambda = 0 \quad \rightarrow \sigma_i^2 + P_i = -\frac{1}{\lambda} \forall i \tag{G.4}$$

$$P_1 + P_2 + ..., P_{N_T} = P_T$$

where we introduced as in the main text the equivalent channel noise variance $\sigma_i^2 \triangleq \sigma_w^2/d_i^2$. Let us now sum term by term all the N_T equations for all channels i's in (G.4):

$$\sum_{i=1}^{N_T} \sigma_i^2 + \sum_{i=1}^{N_T} P_i = -N_T/\lambda \rightarrow -\frac{1}{\lambda} = \frac{\sigma_T^2 + P_T}{N_T} \triangleq \bar{P} \tag{G.5}$$

so that the power allocation equation (G.4) eventually reads

$$P_i = \bar{P} - \sigma_i^2 \tag{G.6}$$

as in the main text.

REFERENCES

1. M. Luise, G. Vitetta, *"Teoria dei Segnali"*, 4a Edizione, Mc Graw Hill International Editions, 2025.
2. B. Porat, *"A Course in Digital Signal Processing"*, Wiley, 1996.
3. A. Papoulis, S. Pillai *"Probability, Random Variables and Stochastic Processes"*, 4a Edizione, Mc Graw Hill International Editions, 2002.
4. G.H. Golub, C.F. van Loan, *"Matrix Computations"*, 4a Edizione, JHU Press, 2013.
5. U. Spagnolini, *"Statistical Signal Processing in Engineering"*, Wiley, 2018.
6. J. Proakis, M. Salehi, *"Digital Communications"*, 5th Edition, Mc Graw Hill International Editions, 2018.
7. T.M. Cover, J.A. Thomas, *"Elements of Information Theory"*, 2nd Edition, Wiley, 2006.
8. D.J.C. MacKay, *"Information Theory, Inference, and Learning Algorithms"*, Cambridge University Press, 2003.
9. W. Ryan, S. Lin, *"Channel Codes: Classical and Modern"*, Cambridge University Press, 2009
10. D.J.C. MacKay, *"Fountain codes"*, IEE Proc.-Commun., Vol. 152, No. 6, December 2005
11. E. Arikan, *"Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels"*, IEEE Transactions on Information Theory 55.7 (2009)
12. A. Shokrollahi, M. Luby, *"Raptor Codes"*, Foundations and Trends in Communications and Information Theory Vol. 6, Nos. 3-4 (2009)
13. A.F. Molisch, *"Wireless Communications"*, Wiley, 2011.
14. A. Goldsmith, *"Wireless Communications"*, Cambridge University Press, 2005.
15. J.H. Holmes, *"Spread Spectrum Systems for GNSS and Wireless Communications"*, Artech House, 2007

16. S.K. Sharma, *"Characterization and modeling of MIMO wireless channels based on correlation tensor"*, Computers and Mathematics with Applications 64 (2012).
17. C. Balanis, *"Antenna theory analysis and design"*, 2nd Edition, Wiley, 1997.
18. B.E.A. Saleh, M.C. Teich, *"Fundamentals of Photonics"*, 3rd Edition, Wiley, 2019.
19. G.P. Agrawal, *"Fiber-Optic Communication Systems"*, 5th Edition, Wiley, 2021.