Project: Performance Analysis of a realistic Cloud Radio Access Network (C-RAN)

Elie Awwad, Philippe Ciblat, Jean-Christophe Cousin, Chadi Jabbour, Kyriaki Niotaki, Germain Pham, Ghaya Rekaya-Ben Othman, Cédric Ware

2024-02-12 version 1.7

Contents

1 General overview

You will study information transmission between connected devices and a remote server through a Cloud Radio Access Network (C-RAN), which combines radio for access and optical fiber afterwards. The main framework will be the development of a Matlab$^{®}$ simulator that models communication from the information bits to the physical signal; you will develop it, progressively deepening the model's complexity until you reach a realistic level, at which point you will then have to satisfy performance requirements that will be detailed in the deliverables below (chapter 2 on page 4). Furthermore, parts of the transmission chain will be studied on actual hardware, which will let you estimate calibration parameters for the simulator.

1.1 Learning objectives

The objective of the project is *not* the simulator in itself, but that you acquire the following knowledge and skills:

- Design an end-to-end communication system against performance requirements (such as error rates) taking into account realistic models of physical signal transmission (in electrical, radio and optical forms).
- Explain the mathematical techniques and models used for data communication in general (modulation, coding) and as a physical signal over specific devices (radio channel, optical fiber...)
- Implement a computer simulator of transmission chains.
- Work in small teams on a software project following best practices (individual deliverables and deadlines, code maintainability, clear presentation of results...)

1.2 System model

We will consider a Cloud Radio Access Network (C-RAN) composed by a set of nodes that may transmit information wirelessly to a basestation. This basestation is connected via an optical fiber to a remote server which corresponds to the final destination as described in Fig. 1.1.

In the project, we will consider that:

- the nodes have imperfect hardware devices;
- the wireless channels are modeled according to realistic configurations;
- the basestation may have hardware devices with flaws or limitations, and may or may not decode the information before forwarding it into the optical fiber;
- the optical transmission over fiber may use either an intensity-modulated or a coherent system;

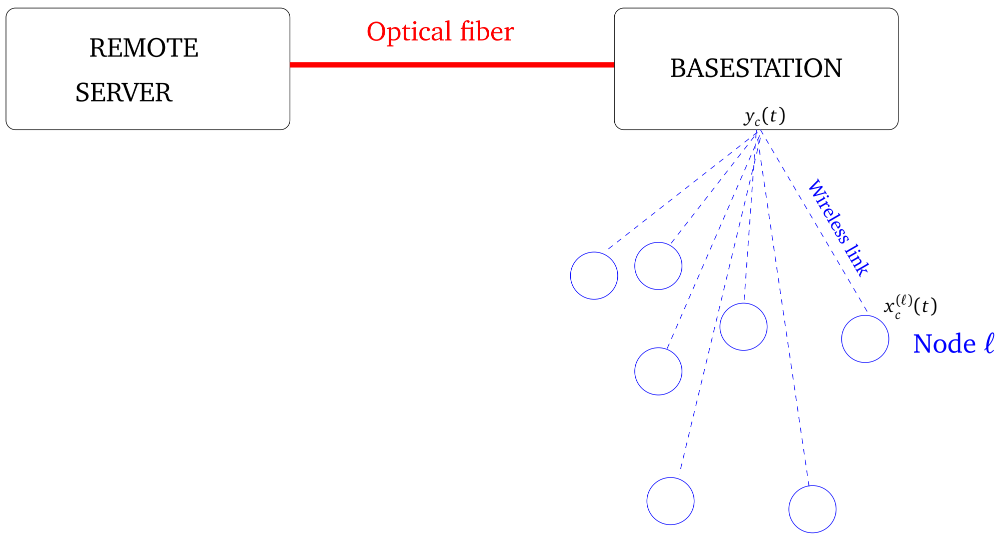


Figure 1.1: *C-RAN system model*

- the final server decodes the information.

The following sections will detail the modeling of these system parts and the C-RAN system as a whole. In terms of notations, we have

- $x_c^{(\ell)}(t)$ the carrier transmitted signal at node ℓ, $x^{(\ell)}(t)$ the baseband corresponding signal.
- $y_c(t)$ the carrier received signal at the basestation, $y(t)$ the baseband corresponding signal.

1.3 Wireless channel modeling

In this section, we introduce the model for one wireless link between a node and the basestation. We thus remove the node index ℓ.

We consider in baseband a multipath channel with M paths, each of whose complex-valued attenuation is A_k and the delay is τ_k.

Therefore we have

$$y(t) = \sum_{k=1}^{M} A_k x(t - \tau_k).$$

The channel described by $\{A_k, \tau_k\}_{k=1,\cdots,M}$ may be time-varying frame by frame where the frame size is N symbols.

Moreover, we assume throughout the project that the channels $\{A_k, \tau_k\}_{k=1,\cdots,M}$ are known at the basestation side and hence no estimation is required.

2 Deliverables

For all deliverables, unless otherwise indicated, you will be expected to produce a short report detailing the results of your assigned studies; and demonstrate the programs that you used to obtain said results. Both must conform to the guidelines and good practices outlined in chapter 4.

Most deliverables have both minimum requirements and extra-credit questions. While you are not required to address the latter in each deliverable, you are expected to do at least some of the extra credit over the whole project. Grading rules are specified section 3.2.

D1: Simulation of system with ideal hardware

We model here the transmit and receive nodes. We consider single-carrier transmission. We first consider an ideal case where hardware devices are perfect and hence not considered.

D1.1 Ideal simulations

We consider the following model given by Fig. 2.1.

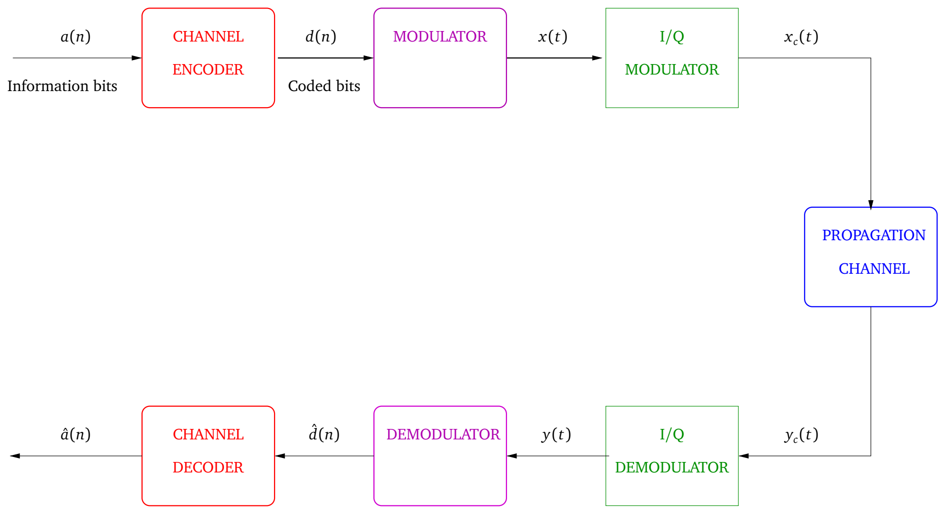


Figure 2.1: *Ideal Transmitter and Receiver*

At the transmitter side, we consider that

- the channel coding is a linear block coding described just below whose input is the information bits slotted $a(n)$ of size K_c and the output is a binary sequence $d(n)$ of size N_c. This binary sequence corresponds to the data to be sent. The considered codes are binary BCH codes of length $N_c = 31$. We will implement two codes:
  - the first BCH code corrects 1 transmission error ($t = 1$),

- and the second code corrects 2 transmission errors ($t = 2$).

  The standard form will be implemented to alleviate the decoding. We note that $GF(32) = GF(2)[x]/1 + x^2 + x^5$. The BCH codes have 2 different rates, which will allow to change the spectral efficiency of the transmission scheme.

- the modulation is linear which means that

$$x(t) = \sum_n s(n) g(t - nT_s)$$

  where

  - $s(n)$ are the symbols obtained via the binary sequence $d(n)$. The symbols may belong to any constellation (PAM, PSK, QAM).
  - T_s is the symbol period
  - $g(t)$ is the shaping filter which satisfies the square-root Nyquist condition.

At the receiver side,

- the received signal in baseband is

$$y(t) = \sum_n s(n) p(t - nT_s) + v(t)$$

  where

  - $p(t)$ is the channel encompassing the shaping filter and the propagation channel,

$$p(t) = \sum_{k=0}^{M} A_k g(t - \tau_k)$$

  - $v(t)$ is the white Gaussian noise

- we apply the matched filter related to $g(t)$ only (in order to simplify). Consequently, we have

$$\tilde{z}(t) = \sum_n s(n) \tilde{h}(t - nT_s) + \tilde{v}(t)$$

  where

  - $\tilde{h}(t) = \overline{g(-t)} \star p(t)$
  - $\tilde{v}(t)$ the colored noise with $\tilde{v}(t) = \overline{g(-t)} \star v(t)$.

- we then sample at the symbol rate $1/T_s$, which leads to

$$z(n) = \sum_{\ell=-L}^{L} h(\ell) s(n - \ell) + w(n) \qquad (2.1)$$

  with $z(n) = \tilde{z}(nT_s)$, $h(n) = \tilde{h}(nT_s)$ and $w(n) = \tilde{v}(nT_s)$ a white noise with zero-mean and variance N_0 (per complex dimension). In Eq. (2.1), we consider that the global filter $h(n)$ has non-zero values over $2L + 1$ symbol times.

Therefore if perfect hardware devices are used, we can assume that Eq. (2.1) holds. As InterSymbol Interference occurs, equalizers have to be employed.

In order to analyze the digital communications transmitters and receivers, we rewrite Eq. (2.1) by using the following matrix framework

$$\mathbf{z} = \mathbf{H}\mathbf{s} + \mathbf{w} \tag{2.2}$$

where

- $\mathbf{z} = [z(N), \ldots, z(1)]^{\mathrm{T}}$ with N being the frame size. To avoid Inter-Frame Interference, each frame is pre-fixed and post-fixed by a null sequence whose length is tailored according to the filter memory length. The superscript $(\bullet)^{\mathrm{T}}$ stands for the transposition.
- $\mathbf{s} = [s(N), \ldots, s(1)]^{\mathrm{T}}$ the data frame.
- $\mathbf{H}$ is a $N \times N$ Toeplitz matrix described as follows

$$\mathbf{H} = \begin{bmatrix} h(0) & h(1) & \ldots & h(L) & 0 & \ldots & 0 \\ h(-1) & h(0) & h(1) & \ldots & h(L) & 0 & \ldots \\ & & & & & & \\ \ldots & 0 & h(-L) & \ldots & h(-1) & h(0) & h(1) \\ 0 & \ldots & 0 & h(-L) & \ldots & h(-1) & h(0) \end{bmatrix}.$$

- $\mathbf{w} = [w(N), \ldots, w(1)]^{\mathrm{T}}$ the white Gaussian noise

We now apply various detectors:

- the threshold detector,
- the Zero-Forcing (ZF) equalizer,
- and the Decision-Feedback Equalizer (DFE).

The work to be done associated with this section D1.1 as well as the numerical values of each of the above-introduced parameters are described in the next section D1.2.

D1.2 Specific deliverable objectives

Coding part

- After calculating the generator polynomial of the considered BCH codes, deduce the rate of the code.
- Then implement the standard coding operation (systematic shape), using the coding method provided in the Appendix.
- Implement an optimal ML decoding for both codes.
- Plot the Bit Error Rate (BER) versus E_b/N_0 for the uncoded BPSK case and both BCH codes (modulated with BSPK) with an AWGN channel. The range of E_b/N_0 could be set between 0 and 10dB. Compare the empirical coding gains with the theoretical ones. We remind that E_b is the energy consumed to send one information bit.

Modulation part We consider three different multipath channels for the communication in addition to the AWGN (denoted hereafter by Channel 0). Once one channel is selected, it is applied over all the frames. We remind that the transmission is done through frames of size N and we consider $N = 100$.

- Channel 0: $\mathbf{A} = [1]$, $\boldsymbol{\tau} = [0] \times T_s$, $(M = 0)$,
- Channel 1: $\mathbf{A} = [1, 0.1, 0.1, 0.1, 0.1]$, $\boldsymbol{\tau} = [0, 0.5, 1, 1.5, 2] \times T_s$, $(M = 4)$,
- Channel 2: $\mathbf{A} = [1, 0.8, 0.6, 0.4, 0.2]$, $\boldsymbol{\tau} = [0, 0.5, 1, 1.5, 2] \times T_s$, $(M = 4)$,
- Channel 3: $\mathbf{A} = [1, 0.8, 0.8, 0.8, 0.8]$, $\boldsymbol{\tau} = [0, 0.5, 1, 1.5, 2] \times T_s$, $(M = 4)$,

We remind that the global continuous-time filter $\tilde{h}$ is given by

$$\tilde{h}(t) = \sum_{k=0}^{M} A_k \tilde{g}(t - \tau_k)$$

where $\tilde{g}(t) = \overline{g(-t)} \star g(t)$ is a Nyquist filter, and more precisely, a raised cosine with roll-off $\rho = 0.5$.

In practice, according to Eqs. (2.1)-(2.2), we only need to implement the discrete-time version of $\tilde{h}$, denoted by h, and given by

$$h(m) = \tilde{h}(mT_s) = \sum_{k=0}^{M} A_k \tilde{g}(mT_s - \tau_k).$$

Finally, h is normalized so that $\sum_{m=-L}^{L} \|h(m)\|^2 = 1$.

- Display h for channel 1, channel 2, and channel 3. We consider that $T_s = 0.05\mu$s.
- Plot the BER versus E_b/N_0 for each channel with three different equalizers (threshold, ZF, DFE) when BPSK is used without channel coding. Typically consider a range of E_b/N_0 between 0 and 20dB.
- Plot the BER versus E_b/N_0 for each channel with three different equalizers (threshold, ZF, DFE) when 8-QAM is used without channel coding. Typically consider a range of E_b/N_0 between 0 and 30dB.
- Plot the BER versus E_b/N_0 for each channel with three different equalizers (threshold, ZF, DFE) when 16-QAM is used without channel coding. Typically consider a range of E_b/N_0 between 0 and 30dB.

Extra credits

- Apply the coding schemes into the frequency-selective channel configurations given by the three above-mentioned channels. Once again, the channel is kept constant over all frames. More precisely, plot and compare the BER versus E_b/N_0 when BPSK is employed with both coding schemes and without a coding scheme.

- Apply an Automatic Request (ARQ) scheme with at most 1 retransmission. A retransmission is run for the frame if this frame has not been correctly decoded previously. A frame is not correctly decoded if at least one bit is in error. As figure of merit, we consider the throughput, i.e. the number of information bits belonging to correctly decoded frames over the number of channel uses (or equivalently, the number of transmitted symbols). If channel coding is used, we consider the BCH with $t = 2$. Here we will consider the E_s/N_0 (instead of E_b/N_0) where E_s is the average energy for the symbol (whatever the nature of the symbol is: uncoded, coded, or retransmitted).

  Plot the throughput versus E_s/N_0 for the following Modulation and Coding Schemes (MCS): BPSK+BCH, BPSK, 8-QAM+BCH, 8-QAM, 16-QAM+BCH, and 16-QAM. For all the simulations, use the channel 0 (AWGN) with a threshold receiver.

D2: Radio-frequency system and hardware modeling

D2.1 Hardware impairments

The purpose of this section of the project is to explore key baseband and RF analog impairments in a transceiver and to demonstrate how to model their impact from a communication system design point of view. The ultimate purpose is to optimize a mixed-signal platform by taking into account the characteristics of the RF/analog front-end. D1 is focused on the channel en/decoder, mo/demodulator blocks and equivalent channel effect (cf. Fig. 2.1). Here, in D2, we focus on the I/Q mo/demodulator blocks with the propagation channel. You can see an expansion of those blocks in Fig. 2.2. This figure also introduces the different electronics blocks and domains that are used in this part of the project.

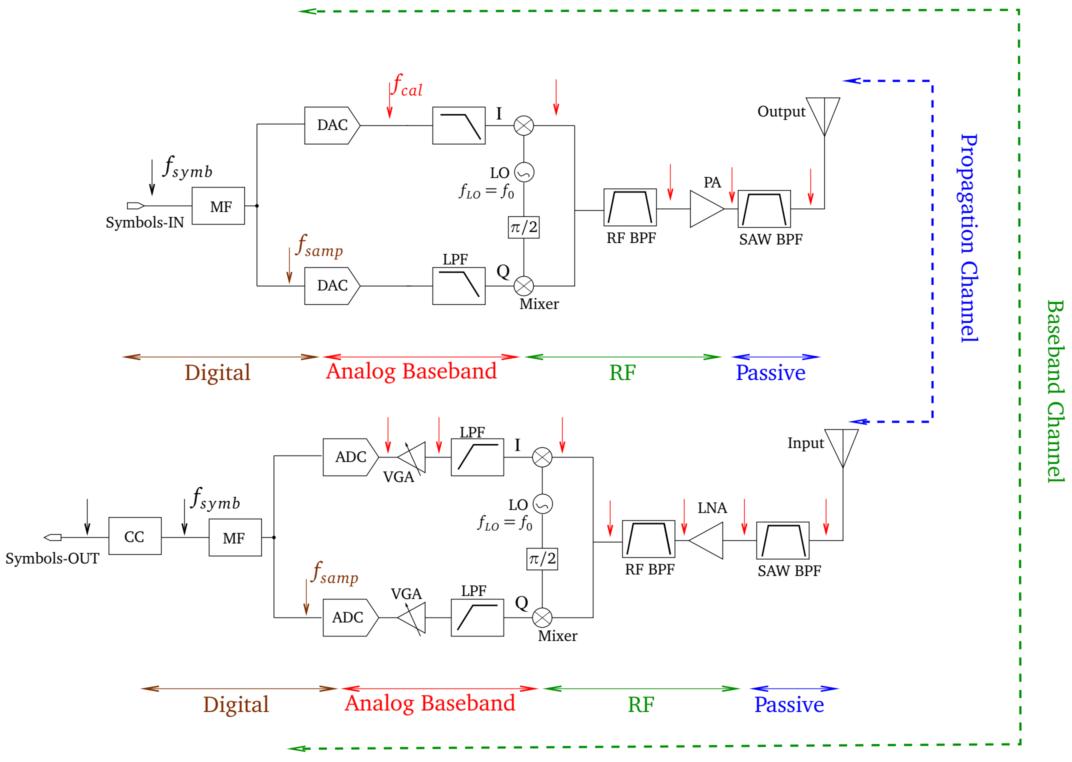


Figure 2.2: *Full transmission chain*

There are two main non-idealities in electronic systems that degrade their resolution and precision: noise and non linearity. It is worth noting that there are many other non-idealities that exist in practice such as process variations and linear distortions but during this project, we will limit our studies to noise and non linearity.

The Table 2.1 gives the general specifications of the system under optimization. It provides some general system parameters and also imposes some performance contraints that your system should meet. You may notice that, when applicable, the parameters are consistent with the D1 part (useful bandwidth, transmission distance...)

RF Bandwidth	30 MHz
Useful Bandwidth	20 MHz
Central Frequency	2400 MHz
SNR receiver output	10 dB
Pout TX	20 dBm
Minimum distance	1.4 m
Maximum distance	1400 m
Matching resistance	50 Ω
Temperature	290 K
ACPR Tx output	45 dB
ADC full scale	±1 V
DAC full scale	±1 V

Table 2.1: *General specifications of the system*

D2.1.1 Non-Idealities

Noise: Noise is a random or a pseudo random signal that could be the result of a physical phenomenon such as thermal agitation (thermal noise) or of a systematic operation that leads to a pseudo-random error such as quantization. Most noise sources are white, i.e., have a constant power spectral density over the bandwidth of interest; but some, such as *flicker noise*, are “colored”, i.e. they have a non-flat frequency response. Noise can affect communication systems anywhere but for analysis purposes, noise impact will modeled by adding a random perturbation to the input signal.

Non linearity: In a linear system, the change of the output is strictly proportional to the change of the input. The huge majority of electronic systems are designed to be linear, however many imperfections make them in practice non-linear. The causes of non linearities are diverse, e.g., transistor’s non-linear behavior, slew rate (due to current limitations), unwanted signal mixing between circuits...A large number of non-linear models exist in the literature, the most complete ones are called memory aware, i.e. the output does not only depend on the current value of the input but also on its past values. Among most employed memory aware models, we can cite the Volterra series and the memory polynomial models. In the scope of this project, for the sake of simplicity, we will limit our modeling to memoryless models such as the polynomial model described in Eq. (2.3):

$$y(t) = \sum_{k=1}^{K_{NL}} \alpha_k x(t)^k, \tag{2.3}$$

where K_{NL} is the non-linearity order, α_k the k^{th}-order non-linearity coefficient, $y(t)$ the output signal and $x(t)$ the input signal. Note that this equation is given for real signals but similar expression will be used during the project for complex signals.
There are three main consequences of non-linearity: unwanted harmonics, inter-modulation products and gain compression. They were studied at length in the scope of course TELECOM201.

D2.1.2 Metrics

To measure the impact of noise and non-linearity on the performance of an electronic system, a large number of metrics could be used. A selection of the fundamental ones are introduced hereafter (few additional metrics will be introduced during labs):

Signal to noise ratio (SNR): It measures the ratio between signal power and noise power. It is usually expressed in dB:

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{Signal}}{P_{Noise}}\right). \tag{2.4}$$

Signal to noise and distortion ratio (SNDR): It is similar to SNR but takes into account distortions also.

$$SNDR_{dB} = 10\log_{10}\left(\frac{P_{Signal}}{P_{Noise} + P_{Distortions}}\right). \tag{2.5}$$

The $SNDR$ can be easily computed from time-domain simulations or from measurements. Two approaches are available according to the knowledge of the input signal:

- Time domain: Any deterministic signal could be used as an input, the $SNDR$ is then given by the following expression

$$SNDR_{dB} = 10\log_{10}\left(\frac{\sum\limits_{n=1}^{N} x[n]^2}{\sum\limits_{n=1}^{N}(x[n] - y[n])^2}\right),$$

  where N is the number of samples of observation.

- Frequency domain: A sinewave should be used as an input signal, this allows to distinguish the useful signal from the noise and non linearity in the output spectrum. The $SNDR$ is then given by the following expression

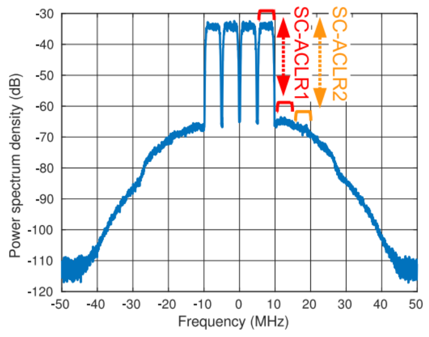
$$SNDR_{dB} = 10\log_{10}\left(\frac{PSD(f_{in})}{\int_B PSD(f)df - PSD(f_{in})}\right),$$

  where f_{in} is the frequency of the input sinewave, B the band of interest and PSD the power spectral density.

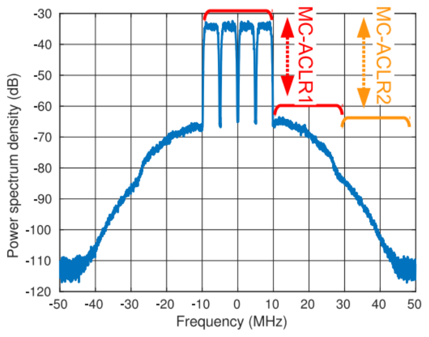
Adjacent Channel Power Ratio (ACPR): When a transmitter is emitting on a given channel, power is also emitted or leaked on adjacent channels due to non-linearity and noise. The ACPR[1] is a metric that measures this leakage. This measure is important to make sure that the degradation to adjacent channels' users is lower than the value fixed in the communication standard. We can compute the ACPR from the power of the main useful channel and the power of the unwanted signal that is in the adjacent channel by the following equation:

$$ACPR = 10\log_{10}\left(\frac{P_{main\ channel}}{P_{adjacent\ channel}}\right) \tag{2.6}$$

A graphical representation of different ACLR definitions is shown in Fig. 2.3.



(a) Single-Carrier



(b) Multi-Carrier

Figure 2.3: *Representation of ACPR computation*

D2.1.3 Operation and sizing

An appropriate sizing and design of the RF chains are mandatory in order to guarantee the signal integrity at the transmitter output and at the receiver output. Each of the blocks of the transmit and receive chains have a specific function but due to noise and/or nonlinearity they will degrade the quality of the signal (which can be measured with the SNDR). The Table 2.2 describes each block of the transmit and receive chains along with their main characteristics. Please refer to Fig. 2.2 while browsing the items of the list. The main target of this project part is to study the effects of those non-idealities and find the right design trade-offs to meet the general system specifications given earlier in Table. 2.1.

D2.1.4 Main structure of this project part

As mentionned earlier, this project part is focused on the hardware electronics of wireless transceivers. Therefore you will have to work on actual hardware boards[2] and on the simulation of hardware electronics.

Hardware focused part: The hardware part is focused on the AD-FMCOMMS3-EBZ Board[1] and on the Pluto Board[2]. Those SDR[3] boards are *small* power transceivers and must be completed

[1] a.k.a. Adjacent Channel Leakage Ratio (ACLR)
[2] The hardware part will be adapted according to health constraints.
[3] Software-defined radio

Block	Function	Main non idealities	Specification
Transmitter			
Digital to Analog Converter (DAC)	Converts the signal from digital to analog	Quantization Noise, Thermal Noise, nonlinearity due to mismatch	Number of bits, fs, Full Scale
Reconstruction filter	Filter DAC replicas to avoid poluting adjacent chan.	Thermal Noise	Order, Bw Filter approximation
Up-Mixer	Up-mixes the signal around the LO frequency	Phase noise, nonlinearity	IIP2
Power Amplifier (PA)	Amplifies the signal before transmition	Non linearity	Gain, P1dB, IIP3
Receiver			
Low Noise Amplifier (LNA)	Amplifies the signal at the receiver input	Noise	Gain, noise figure
Down-Mixer	Down-mixes the signal from the LO frequency to DC	Phase noise, nonlinearity	IIP2
Anti-Alias filter (AAF)	Filter interferers before the sampling to avoid aliasing	Thermal Noise	Order, Bw Filter approximation
Baseband Gain	Amplifies the signal to occupy the full scale of the ADC	Thermal noise	Gain, Noise Figure
Analog to digital Converter (ADC)	Converts the signal from analog to digital	Quantization noise, Thermal Noise	Number of bits, fs, Full Scale

Table 2.2: *Blocks of the transmit and receive chains with their main characteristics*

by power amplifiers (PAs) modules for *long distance* transmissions.
The main targets regarding the hardware part are:

- fully understand the SDR hardware structures
- understand the process of PA characterization

Simulation focused part: The simulation part is focused on the implementation of behavioral models taking into account some hardware impairments and on the analysis of their effects on the system. Most of the blocks are actually already implemented and your main task is to understand how to use them in your simulations.

The strategy is first, to study each functional block independently with in-depth analysis and make some preleminary design choices. This phase can be refered to as *unitary simulation tests (UST)* and each project lab session is dedicated to a block of the chain.
The main purpose of these sessions is to unlock you on the analysis of these blocks rather than offer you a presentation. **Therefore, you must acquire the basic understanding of the blocks by yourself before the project lab sessions**. The second step is to connect the blocks together and check the overall performance. The final step is to perform global simulations.

Depending on the possibility of carrying out hardware measurements, you may even be able to enhance the simulator with measurements results.

D2.2 Specific deliverable objectives

The expected deliverables result from the work carried out on each sub-part.

- Hardware focused part:
  - Plot the measured AM/AM and efficiency characteristics of the PA allocated to your group
  - Perform a SNR vs distance measurement using the SDR platform
- Simulation focused part:
  - For each functional block (UST), provide the detail performance analysis for each case: transmitter and receiver. Example:
    - filters: passband and stopband frequency and attenuations analysis,
    - converters: quantization noise and sampling frequency analysis,
    - ...
  - Provide a hardware configuration that minimises the RX and TX power consumption that meets the desired specifications
  - For the global simulation chain, provide the SNR at the receiver output (ADC) against the distance between transmitter and receiver.

Extra credits:

- Hardware focused part:
  - Provide a description of each blocks of the SDR board assigned to you with the support of slides and explain the hardware main specifications of the blocks
  - Provide an analytical fitting of the attenuation with respect to distance and frequency
- Simulation focused part:
  - Plot the BER against the distance between transmitter and receiver.
  - Use one of the en/decoder of D1 to encode and decode the bitstream,

D3: Optical point-to-point system performance

We assume that the basestation forwards aggregated information to a remote server in the cloud, as light over an optical fiber. The information forwarded may be either the users' data after decoding it from the radio channel (decode-and-forward architecture), or the raw radio waveform (quantize-and-forward architecture). The latter saves cost on the RF hardware and signal processing at the basestation, but puts a heavier strain on the optical communication system. Since this is part of the access network, where operating and capital expenses (OpEx and CapEx) are shared by fewer users, the need to keep costs low is emphasized. Must you pay for a heftier optical system, or can you improve your digital communications techniques to accomodate a cheaper one, without losing on processing costs?

D3.1 Issues with optical fiber

We can assume that the fiber used is a standard singlemode fiber according to the G.652 standard. The main physical impairments for a signal traveling over such a fiber are:

- Attenuation: the signal loses power exponentially with propagation length. The minimum attenuation is about 0.2 dB/km, for a signal in the C band (carrier frequency around 193 ± 3 THz, corresponding roughly to $1530-1560$ nm in wavelength); attenuation will be higher outside that range. You will **experimentally measure** the attenuation coefficient of a fiber through an optical time domain reflectometry (OTDR) setup, then use the measured value in your simulation.
- Dispersion: a quasi-monochromatic wave propagates as $e^{i(\omega t-\beta z)}$, but β varies with ω, and is usually approximated by a second- or third-order Taylor series around the carrier frequency:

  $$\beta(\omega) = \beta_0 + \beta_1(\omega-\omega_0) + \frac{1}{2}\beta_2(\omega-\omega_0)^2 + \frac{1}{6}\beta_3(\omega-\omega_0)^3 + \dots$$

  You will **experimentally measure** β_2 or equivalently the dispersion coefficient D over a standard fiber in the C band and use the measured value in your simulation. Dispersion can be alleviated in the O band (around 1300 nm), where it is lowest, at the cost of a higher attenuation.
- Polarization: the direction of the optical field is not conserved in the optical fiber due to random stressing and twisting. This is not a major issue with systems based on intensity modulation and direct detection (see below) but requires polarization-diversity coherent detection systems when modulating the amplitude, phase and state-of-polarization of the optical field.
- Other physical impairments, chiefly nonlinearity (Kerr effects), can be neglected for access networks (short distances) and even metropolitan networks; but they can be considered as a limit on the maximum optical power that can be injected into the fiber: about 10 dBm in the C band; or 0 dBm in the O band.

D3.2 Issues with optical transmission systems and devices

Apart from the optical fiber itself, the optical or opto-electronic elements used in a transmission system are lasers, modulators, photoreceivers and filters or demultiplexers. Depending on their arrangement, they can be classified into the following categories:

- Laser: converts an electrical current into optical power. Semiconductor lasers typically have a static characteristic $P(I)$ with a threshold of $I_{\mathrm{th}} \simeq 20$ mA, above which it is linear up to about 5 times I_{th}, reaching powers up to about 10 dBm. In dynamic regimes, the response has a resonant frequency of around 10 GHz, variable with the DC current. (See TELECOM203 lecture and tutorial on semiconductor lasers.) In addition, lasers emit intensity and phase noises, and their optical frequency varies with the bias current, temperature and other external factors. You will measure some of the laser characteristics through a dedicated lab session.

- Direct modulation: information is imparted onto the optical signal by modulating the bias current of a laser. This solution is a very simple and cheap implementation of intensity modulation. However, a phase and frequency modulation is also imparted; and at modulation frequencies in excess of about 1 GHz, the impulse response of the laser must be taken into account.

- External modulation: information is imparted onto the light from a CW laser by a separate voltage-controlled modulator. Different modulator types operate on optical intensity, phase or both. Typical bandwidths are on the order of 20–50 GHz, though power amplifiers are usually required to produce enough voltage at frequencies above a few GHz.

- Photoreceiver, direct detection: converts optical power into electrical current, with a typical sensitivity of 1 A/W at 1550 nm. Adds a white noise roughly proportional to the temperature (thermal noise).

- Coherent detection: allows access to the optical field's complex amplitude, instead of only the intensity. A local laser (or local oscillator, LO) is used to interfere with the received signal. If the optical field is $Ae^{i\phi}$, then 2 interference paths with different phase shifts sent to a pair of balanced photodetectors yield $A\cos\phi$ and $A\sin\phi$.
  A full polarization-diversity implementation of coherent detection also requires 2 polarization channels, for a total of 8 photodetectors. In theory, this also requires the optical frequency of the LO to be locked to the signal's carrier frequency; this is exceedingly hard to ensure, so in practice the LO is left unlocked, which results in an $e^{i\Omega t}$ phase shift on the signal with a random frequency offset Ω. Signal processing techniques allow to track this phase shift up to a few 100 MHz.
  You will **measure the sensitivity gain** that coherent detection offers with respect to direct-detection through a simplified experimental setup by measuring the signal-to-noise ratio for the two detection schemes.

- Optical filters and demultiplexers: allow parts of the optical spectrum to be blocked or separated. Unlike their electronic counterparts, they can be assumed to have a typical insertion loss of about 5 dB, bandwidths greater than a few nm (a few 100 GHz), and slopes no sharper than 20 dB/nm, at least for devices cheap enough to be used in access networks.

D3.3 Specific deliverable objectives

- **Focus point 1 - Lab work 1:** Measure the attenuation and dispersion coefficient of a standard singlemode optical fiber. See the specific lab work description to perform the measurements. The results are to be used in the simulation parameters.

- Implement a simulator function to model propagation in an optical fiber with given attenuation (dB/km) and dispersion (β_2, β_3). You can use the `fft` function to work in the spectral domain. You will determine the conditions under which this is legitimate (e.g. length of null sequences at the beginning and end of the sample vector representing the signal). Check its effects on a short pulse.

- Using the given functions `TX_optical_eml` and `RX_photodetector`, study the BER of an optical system back-to-back (without fiber) using on-off-keying (OOK) intensity modulation and direct detection at $R_b = 2.5$ and 10 Gbit/s. Deduce the required optical powers at the photodetector in each case to get a BER lower than 10^{-3} and 10^{-6}, without equalization nor error correcting codes.

- Model the same system over up to 100 km of fiber. Show that the BER increases after a certain length L_{max}, even if you maintain a constant power at the receiver side (RX).

- Using the given function `TX_optical_dml`, study the performance of a direct modulation of the laser with an OOK signal at 1 Gbit/s, back-to-back and over 20 km. More precisely, modify the value of the bias current I_{DC} of the laser and study its impact on the modulation.

- **Focus point 2 - Lab work 2:** Compare the signal-to-noise ratios of the following optical receiver configurations: direct detection and coherent detection. See the specific lab work description to perform the comparison.

- Propose and study solutions to reach 100 km at 10 Gbit/s with an uncorrected BER lower than 10^{-3}. By uncorrected, we mean without the use of forward error correction codes.

Extra credits:

- Study the dependence of L_{max} in R_b. Compare it with a simple analytical model based on how much pulses broaden due to dispersion.

- Implement and study the performance of digital equalization inspired from D1 to equalize the received optical signal before performing threshold detection. To do so, you should first find or estimate the channel response and then deduce an equalizer. Study the performance enhancement by comparing the BER before and after equalization.

D4: Network simulation and global performance

D4.1 Cellular Network

We will first study the performance between the nodes and the basestation in a standard cellular network.
We consider K active users. In order to manage the multiple access between these users, we choose a very simple technique: the so-called Time-Division Multiple Access (TDMA). Each TDMA superframe is divided into K frames, one per user. Each frame is of length N. The communication is done on the bandwidth B and the carrier frequency is denoted by f_0.

For each user, we have a certain Quality of Service (QoS) described as follows:

- The rate of each user ℓ is fixed and denoted by R_ℓ. If needed, we may consider four user modes:
  - $R_\ell = 40$ kbit/s (2G-GPRS/EDGE)
  - $R_\ell = 400$ kbit/s (3G-WCDMA/UMTS)
  - $R_\ell = 4$ Mbit/s (3G-HSPA/H+)
  - $R_\ell = 40$ Mbit/s (4G)
- The Frame Error Rate (FER) –as soon as a bit in a frame is wrong, the entire frame is assumed to be not well detected and discarded–. The access control will reject the user if its FER is greater than a pre-defined threshold $F_{\max}$.
- The energy used for each transmitted symbol, i.e., the symbol energy E_s, is upper-bounded by $E_{\max}$. The access control will reject the user if its symbol energy is greater than $E_{\max}$ in order to satisfy the rate constraint R_ℓ and the FER constraint $F_{\max}$.

In order to compute a network, we need to locate the nodes around the basestation according to a certain model. We will use a uniform distribution of the K users within a square whose side measures $D_{\max}$ and the basestation is located in the center of the square. In practice, the square corresponds to a cell. The different figures that will be defined in D4 have to be averaged over various realizations of the users' locations.

Given one realization of the users' locations, find the corresponding pathloss between each user and the basestation. Given the pathloss, you have to select randomly with a uniform distribution one channel impulse shape out of the three frequency-selective channels described in section D1.2. The channel impulse response h has to be rescaled according to the pathloss given by the Friis equation where the antenna's gains are equal to 1.

D4.2 Cloud Radio Access Network

Now we focus on the whole system as drawn in Fig. 1.1 where the basestation plays a role of relay from the user to the remote server. Then, we apply

- either the Decode and Forward (DF) approach where the basestation attempts to decode all the streams coming from the nodes associated with it (so by decoding the users assigned on this cell). Then these streams are re-encoded (as in section D3.3) to go into the optical fiber.
- or the Quantize and Forward (QF) where the basestation only samples and quantizes the raw received signal and this quantized signal is encoded to go through the optical fiber. The quantification of the received samples is done over q bits/real dimension.

D4.3 Specific deliverable objectives

We define the rejection rate as the number of unserved users due to a constraint outage over the total number of users denoted K. The constraint outage comes from the power outage which occurs when the required power to do the transmission is larger than $P_{\max} = 20$ dBm for ensuring a FER of $F_{\max} = 10^{-2}$.

We consider that all the users use the same mode and the same receiver. And we choose $D_{\max} = 2$ km.

We choose the following set of values: $B = 30$ MHz, including a roll-off factor $\rho = 0.5$ (leading to $T_s = 0.05\ \mu$s), the carrier frequency $f_0 = 2.4$ GHz and $N_0 = -174$ dBm/Hz.

- Compute the rejection rate at the basestation versus K for 8 configurations defined by a mode (one out of the four rate values R_ℓ) and a receiver (one out of the two equalizers: ZF or DFE) by following these steps:
  - To evaluate the presence or absence of a constraint outage, build a Look-Up Table given the mininum required E_s/N_0 (in dB) at the receiver side leading to a FER of $F_{\max}$ for each constellation (BPSK, 8-QAM or 16-QAM), each channel (one of the three models defined in D1), and each receiver (ZF or DFE). To do that, find first a closed-form expression between FER and BER with respect to N and the constellation size.
  - Evaluate the maximum E_s (according to the $P_{\max}$ constraint) and deduce the maximum available E_s/N_0 (in dB) at the transmitter side.
  - For each configuration, select randomly a set of users' locations (for a given K) and find the corresponding pathloss between each user and the base station. Then, select randomly with a uniform distribution one channel impulse response, apply the corresponding pathloss and count the number of users in outage. Average over a lot of realizations. Finally plot the rejection rate versus K. Take K within the interval $[1, K_{\max}]$ where $K_{\max}$ depends on the user mode (for instance, if $R_\ell = 40$ kbit/s, then $K_{\max} = 2000$).
- Calculate the raw data rate required on the optical fiber for each relay approach (DF or QF) with at most $q = 10$. Is the optical fiber the bottleneck of this C-RAN system?

Extra credit We consider that we have two adjacent cells denoted $\mathscr{C}_{-1}$ and $\mathscr{C}_1$, one at each side of the central cell denoted $\mathscr{C}_0$. At a given time slot, we have one active user per cell. Each user operates with a BPSK modulation. For any link, channel 0 is considered however while taking into account the path loss between each user and the basestations.

- For $\mathscr{C}_0$, what is the loss of performance (in terms of BER) if the multi-cell interference is not managed and seen as an additional noise? To do so, plot the BER versus P where $P \in [0, 20]$ dBm.
- We now consider that the three basestations send their samples to the server without errors. Consequently, at each time index, the server has three samples y_{-1}, y_0 and y_1. For this time index, the active user in cell $\mathscr{C}_i$ sends the BPSK symbol s_i.
  - Write the vector $\mathbf{Y} = [y_{-1}, y_0, y_1]^{\mathrm{T}}$ with respect to $\mathbf{S} = [s_{-1}, s_0, s_1]^{\mathrm{T}}$ and the channel losses.
  - At the server side, evaluate the BER for the user assigned to cell $\mathscr{C}_0$ when a ZF receiver based on $\mathbf{Y}$ is carried out.

3 Practical information

3.1 Planned schedule

Semester S2, Period P3	TH1–2	Simulation with ideal hardware
	TH3–4	Simulation with ideal hardware
	Deliverable D1	
	TH5–6	DAC + ADC
	TH7–8	DAC + ADC Filtering at RX and TX
	TH9–10	Mixer at RX and TX
	TH11–12	Power amplifier + low-noise amplifier
	TH13–14	Integration + optimization
	TH15–16	Optics (Project + Lab sessions Telecom 203)
Semester S2, Period P4	TH17–18	Optics (Project + Lab sessions Telecom 203)
	Deliverable D2	
	TH19–20	Optics (Project + Lab sessions Telecom 203)
	TH21–22	Optics (Project + Lab sessions Telecom 203)
	TH23–24	Optics (Project + Lab sessions Telecom 203)
	TH25–26	Optics: direct vs external modulation
	TH27–28	Network and global performance
	Deliverable D3	
	TH29–30	Network and global performance
	TH31–32	Presentation Deliverable D4

Each deliverable is due after two weekends from the last session of each chapter. Exact deadlines are provided on E-campus. You will have to set a meeting with professors in the following days to review your results and code. You may set a meeting with your tutor before or after the deadline to review the deliverable.

3.2 Grading

- 15 % for each deliverable, 20 % for the final presentation, 20 % for at least 4 “extra-credit” questions (see below).
- In general, each deliverable will be graded as:
  - 25 % results successfully obtained,
  - 25 % interpretation and exploitation of the results,
  - 25 % explanation of the models and mathematical techniques used,

  - 25 % quality of the code and of the report.

- Each deliverable includes questions marked “extra credit”, which count towards the extra-credit grade instead of the deliverable itself. A minimum of 4 such questions must be answered over the whole project to get a perfect grade.

3.3 Matlab program

Your simulator will be implemented in Matlab (using the school’s or your own legally-downloaded licenses). The functions and code samples that will be provided to you were tested with Matlab.

Depending on the specific deliverable, you will usually implement simple models for various parts of the system. Then you will be provided with functions for more complex models, or if you didn’t succeed in implementing the simple models correctly but need to move on to the next deliverables (Your grade will of course depend on your results).

4 Appendix: guidelines and good practices

4.1 Good practices: reports

4.1.1 The one rule guide to making good charts

Making good charts is child's play. However, when it actually comes to producing one by ourselves, major failures usually occur[1].

Here is a minimalist guide on good practices for drawing a good chart[2]. We will limit ourselves to the elements that are really essential to a good layout. (The advanced reader who wants to improve on the topic is encouraged to read Loren Shure's excellent "Making Pretty Graphs" posts [3])

If there is a single one idea to remember from this mini-guide, it is "**MAKE LEGENDS**". This applies to the three elements that make up the figure: its **title**, its **axes** and its **plot lines**.

Let's walk you through the step-by-step process of making a good chart.

We have a table of values, derived from measurements or calculations.

```
mois = [1:12];
temp_minimale_paris = [-4.3 9 5.9 9.3 7.4 15.6 14.6 16.7 13.8 13 5 0.8];
temp_minimale_papeete = [25.5 25 26.2 25.3 23.9 23.9 24.3 23 21.7 21.5 25.5 25];
```

We draw the basic graph (the one that must absolutely not shown to a third person because it is unusable!):

```
plot(mois,temp_minimale_paris)
hold on
plot(mois,temp_minimale_papeete)
hold off
```

Now we go to the essential step of this mini guide: **MAKE LEGENDS** ! This includes :

1. To put a title to the chart (ideally that states the goal of the chart)
2. To put a legend for all axes, ie to indicate for each axis the name of the associated quantity as well as its unit, in parenthesis
3. To put a legend for the different lines

Here are the associated command lines:

```
% Title of the graph
title('A Year of minimal temperatures on the 1st of the Month in 2017')
% Axes' names
xlabel('Month number')
ylabel('Temperature (deg. C)')
% Lines' names
legend('Paris (France)','Papeete (Fr. Polynesia)')
```

[1]which can have direct impacts on grades...
[2]which increases the probability of high grades...

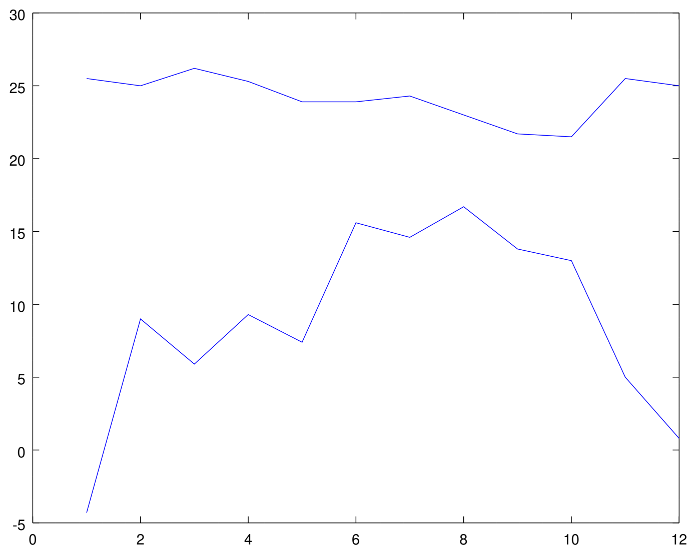


Figure 4.1: *Displayed figure at the screen: example of bad figure*

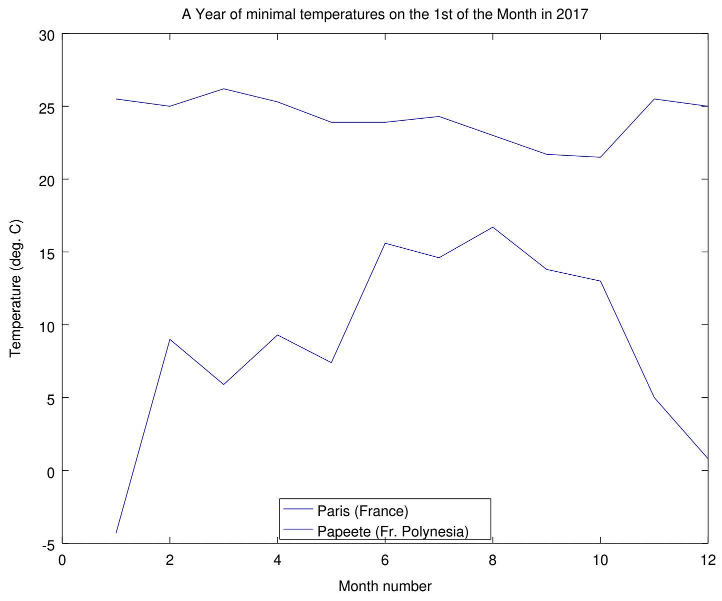


Figure 4.2: *Displayed figure at the screen: example of somewhat better figure*

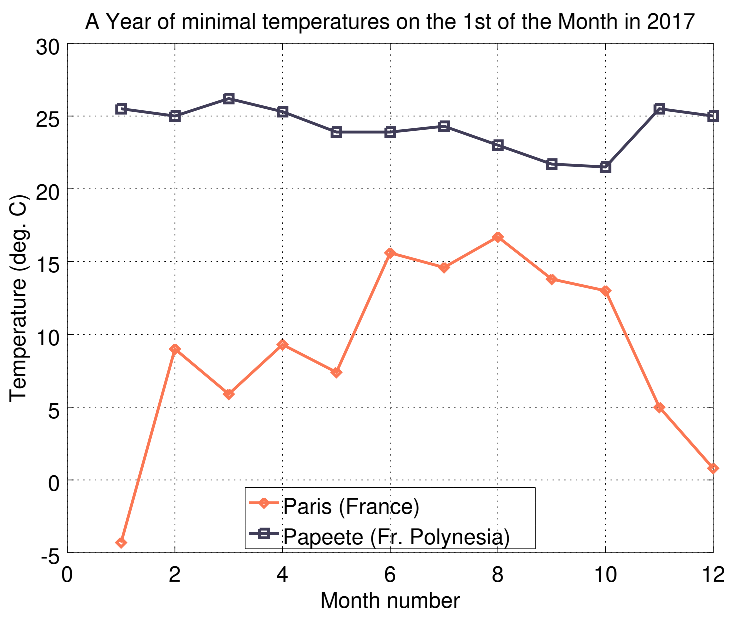


Figure 4.3: *Displayed figure at the screen: example of good figure*

Now we can go to the aesthetic part (which is strongly recommended):

```
% Display grid
grid on
% Get plot handles for each line
line_hdl=get(gca,'children')
% Change markers shape
set(line_hdl,{'marker'},{'s';'d'})
% Change color lines
set(line_hdl,{'color'},{[64,61,88]/256;[252,119,83]/256})
% Change linewidth
set(line_hdl,'linewidth',2)
% Set Fontsize of text elements
set(gca,'fontsize',15)
set(get(gca,'title'),'fontsize',15)
set(get(gca,'xlabel'),'fontsize',15)
set(get(gca,'ylabel'),'fontsize',15)
```

Comment : the last commands for changing the font size can be implemented with the following one-liner command :

```
set(findall(gcf,"-property", 'fontsize'),'fontsize',15)
```

It is advisable to configure your *startup* file so that the font size and the linewidth have the proper default values.

For GNU Octave, the startup file is `~/.octaverc` [3] and default values can be set as documented in: Managing Default Properties (GNU Octave).

For Matlab, the startup file is `startup.m` [4]. The default place for this file is platform-specific[5]:

[3]https://octave.org/doc/v8.4.0/Startup-Files.html
[4]https://fr.mathworks.com/help/matlab/ref/startup.html
[5]https://fr.mathworks.com/help/matlab/ref/userpath.html

- Windows® platforms : `%USERPROFILE%/Documents/MATLAB`
- Mac platforms `$home/Documents/MATLAB`
- Linux® platforms `$home/Documents/MATLAB`

An example of `startup.m` file is given in the following page: Basic Matlab Startup file ($192) · Snippets · GitLab

4.2 Good practices: programming

As with any software development project, following good practices will help ensure that your results are accurate, avoid or solve programming mistakes, and save you time in the long run. Here are some recommended guidelines, further documented in [4, 5].

4.2.1 Use a version control system

Don't share code by email or a mere folder such as our school's Owncloud service, let alone Dropbox®. You *will* get confused between versions. Git repositories will be provided. Use them as the reference version of your code.

See section 4.3 for more details and help. **The homework section 4.3.5 on page 30 details the specific operations you MUST be familiar with** for efficient collaborative work in the project.

4.2.2 Document everything, and keep it up-to-date

If you don't see the need to make your code easy to understand to other people, consider that yourself next month will be another person. You *will* waste time trying to understand what you had in mind when you wrote last month's code if you're not careful. Additionally, your teachers will evaluate your code, and will be in a better mood if they don't have to warp their mind around this undocumented optimization that you thought was so clever while pulling an all-nighter.

Comments in Octave begin with `%`. An whole line of comments usually begins with `%%`. Your text editor/development environment will indent them accordingly (see section 4.2.3). Use them to document:

- Functions: the first few lines of comments in a function are displayed when you type "`help` *<function>*" in Octave. Use this to tell the reader what your function does, both in short and in detail; what are the expected parameters and return values, what they represent and any assumption you make; and for more complex functions, some examples and non-obvious corner cases.
- Blocks of code: you don't have to add a comment to every line of code, nor to write trivialities such as:

  ```
  x = x + 1 % Add 1 to x.
  ```

  Rather, isolate meaningful blocks of code and document what the code achieves.

```
%% Count total symbols sent, print status every million symbols.
symbols_sent = symbols_sent + N;
if (~ mod(symbols_sent, 1e6))
  fprintf('%d sent, %d errors... ', symbols_sent, errors_dfe);
end
```

Finally, keep this documentation religiously up to date. You may waste a lot of time before realizing that a section of the code doesn't do what the comments say, because someone forgot to update them. Always keep them synchronized.

4.2.3 Use spacing and indentation for best legibility

Don't pack your code into the smallest possible space. Make it easy to read. If a formula doesn't fit on a single line, break it into several lines, adding "`...`" at the end of each unfinished line.

Your text editor/development environment will automatically indent code blocks to provide a visual reference of where it fits. This helps not only to read it but also to write it: if the automatic indentation level surprises you, then perhaps you forgot an `end` or a closing parenthesis somewhere.

4.2.4 Don't repeat yourself

Any piece of data or code should have one single representation in the entire program. If you copy-paste values or code, you *will* forget to update all of them when an update is required. Use functions instead, and pass them parameters. You may group related values in a `struct` and pass that as a single parameter.

4.2.5 Use meaningful variable names

It should always be obvious what a variable represents. The common-sense method is to name them `nb_errors` or `nb_symbols_sent` instead of `n_e`, `n_s`. However, in scientific computing, this may clash with the mathematician's habit of giving variables single-letter names. We recommend that single-letter variable names only be used sparingly:

- for a limited number of standard notations shared across the entire program; even then, you may want to index them (as in `X_in`, `X_out`);
- for local variables within a short function.
- Also, beware of loop counters: the traditional notation `i` clashes with Octave's usual notation for $i = \sqrt{-1}$; avoid calling your counter `i`, and/or use the explicit notation `1i` for $\sqrt{-1}$.

4.3 Good practices: version control using Git

4.3.1 Basic Git and version control concepts

This section is intended for people not already familiar with Git. For more details, you may also refer to first-year lectures on the topic [6, 7] (in French, partly based on SmartGIT, a graphical wrapper around Git); or to any of the many tutorials available online.

Version control concepts in general will be expressed in Git command-line terminology; graphical Git tools will have similar concepts (as well as most other version control software).

- A **repository** contains all your project's files, with a full revision history and references to who originated each line of code.
- A **working copy** is a repository in which you'll be editing the files. That's what you'll typically have on your hard disk: a folder containing your files, that you'll be working on, and a hidden subdirectory `.git/` containing the revision history.
- **Cloning** a repository (command: "`git clone`") means creating a working copy with the same files as another repository (usually called "`origin`"), set up to enable synchronization between them. Typically your working copies will be cloned from your main project repository, which is hosted in the school's Gitlab server.
- A **changeset**, also called "**commit**", represents the change between different versions of your project. Git stores the history as a series of changesets.
- **Staging** files (command: "`git add`") means marking the state of your chosen files in your working copy to be part of the next changeset. Git calculates the difference between the selected files' current content and their state in the previous changeset. The previous changeset is called "`HEAD`".
- `git commit` creates a new changeset from the file contents that you selected using `git add`. (Watch out: if you have modified one of the files after `git add`, the new changeset will not contain your modifications, it uses the file contents *at the time you did* "`git add`"!) After the commit, "`HEAD`" is updated to designate this new changeset.
- `git push` uploads your new changesets to the `origin` repository. It will fail if someone else has been working on it in the meantime, therefore **you must always *pull* from the `origin` before pushing** (see below).
- `git fetch` downloads any new changesets from the `origin` repository, but doesn't update the files in your working copy.
- `git merge` (simplified version) resynchronizes the files in your working copy with any changesets downloaded from the `origin`. If they can be automatically merged with your local changesets (no "conflicts", see below), the new file contents are automatically staged; you can then create a new changeset using `git commit` to finalize the merge, and upload it using `git push`. (Note: you must always commit all your changes before doing `git merge`.)
- `git pull` = `git fetch` + `git merge`.
- A **conflict** is a situation where Git fails to merge the local changesets and the `origin`'s. At this point, **your files have been modified, you must fix them** before you can `git push` again (or you can abort the merge using `git merge --abort`, which restores the files to their previous state until you can deal with the problem). That situation can largely be avoided by working with branches as described in sections 4.3.2 and 4.3.3. (Conflicts may still arise when merging different branches, but the situation will be better controlled.)

Fixing conflicts: a number of tools can be used to fix merging conflicts. The most straightforward is editing the files as you would normally do in your editor, looking for sections where Git couldn't decide which version to keep, that will be marked by lines containing <<<<, ==== and >>>>.

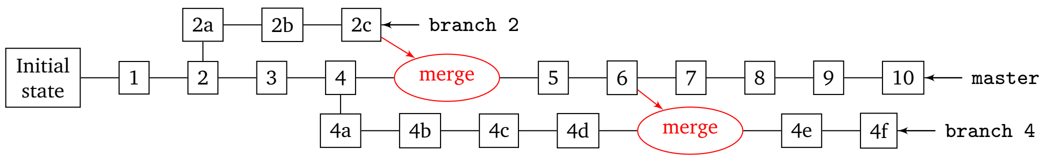


Figure 4.4: *Example Git repository history with multiple branches. Each node represents a changeset. "`master`" is the reference branch. "`branch 2`" diverges from "`master`" at changeset 2, and is later merged into master: typical of a temporary development branch. "`branch 4`" diverges from "`master`" at changeset 4, and merges* from *"`master`" at some point but continues on: typical of an ongoing development branch that sometimes reincorporates other changes from "`master`".*

Otherwise you can see what `git mergetool` does on your system; or maybe you're using a GUI wrapper around Git that integrates a tool to help with conflicts.

Other useful git commands:

- `git status` tells you the state of your working copy: which files have been modified, what branch it is on (see section 4.3.2), and whether there are new changesets to upload or to merge. **Always check `git status` before doing `git commit`!**
- `git diff` shows the differences between the files' current state and what it was in the latest changeset (actually the changeset currently designed `HEAD`).
- `git checkout [`*commit-id*` -- ]` *file* restores a file to the state it had in `HEAD` (or in the changeset *commit-id* if specified), erasing the modifications.
- `git log [--graph]` lists the changesets (and their id!) on the current branch. The `--graph` option shows the merges if any.

If you really can't find a way out: you can always clone your main repository into another working copy, transfer your files from your broken working copy in the state you want them in, then commit in the new working copy and use it instead of the broken one. Normally you shouldn't need to do this: the Git commands described above, and many others, should allow you to recover from mistakes (see in particular [8]); but if you're not familiar with them, it can be quicker to restart from a new clone. Note: you can always recover the local versions of your files using `git log` and `git checkout`.

4.3.2 Collaborative work with Git branches

Your project's history doesn't have to be linear. All changesets are relative to a prior version, but multiple "branches" can diverge and be merged later. Figure 4.4 shows an example with 2 branches in addition to the `master` branch.

You will use your repository's `master` branch as the reference version of your code; and **use branches to experiment with code without breaking the reference version**. After experimenting, you can merge the features you developed in the experimental branches back into `master` using `git merge`, as per the workflow described in section 4.3.3.

- `git switch [-c] "`*branch name*`"` switches the working copy to the given branch.[6] Use `-c` to create the branch if it doesn't already exist.
- Git tracks branches both locally in your working copy and in your `origin` repository. Note the difference between `master` and `origin/master` (or `origin/`*any branch*) before and after `git pull`.
- `git merge "`*branch name*`"` merges the given branch into the current branch. (Without a branch name, as noted above, the default behavior merges `origin/`*current branch* into the current branch.) As always, you may have to solve conflicts as described above.
- `git commit` and `git push`, as noted above, create and upload new changesets *in the current branch*. You may get an error if you try to `git push` but created the branch only locally in your working copy (which is the default); the error message will tell you what command to execute to create the branch also in your remote repository (with the `--set-origin` option).

4.3.3 Recommended workflow

Once per project:

- `git clone` to get a working copy into a folder on your local computer. Go into the folder.
- `git config user.email "`*your email*`"` and `git config user.name "`*your name*`"` to identify yourself as the author of the work you're going to do. (Alternatively, you can use `git config --global` to do this step once and for all the Git repositories you will use on that computer unless otherwise indicated.)

For every session:

- Go into your working copy.
- `git switch -c develop-`*name-yyyymmdd* to create and switch to a new development branch named "`develop-`*name-etc.*" dated from today (should be a unique name).
- Work on the files in the working copy.
- For every consistent change:
  - `git status`, `git diff` to check which files have been modified and how;
  - `git add` to stage the relevant files;
  - `git commit` to create the changeset (remember to write a meaningful description);
  - (better err on the side of smaller changesets with fewer files).
- At the end, either `git push` your local branch to the main repository, if you just want your work to stay in that branch, or merge it into the `master` branch:
  - `git status`, make sure all your changes are committed;
  - `git switch master` to go back to the `master` branch;

[6] Older Git versions used `git checkout [-b]` before the introduction of `git switch`. It can still be used, but may lead to ambiguities. **Do not give a branch the same name as a file!**

  - `git pull`, `git log` to get and review the latest changes to `master`;
  - `git merge develop-`*name-yyyymmdd* to incorporate your work into the branch;
  - if there are conflicts, fix them, then `git status`, `git commit` to create the changeset that completes the merge;
  - (alternatively, `git merge --abort` to return the files to their state in `master` if you don't want to fix the conflicts for now;)
  - after the conflicts are fixed and the merge is complete, `git push` to send your changes to the main repository.

4.3.4 Other version control tips and tricks

- If there is an error while using Git, pay attention to the error message. Git is more informative than most software, tries to guess what you're trying to do, and gives you the necessary command.
- The command `git stash` automatically saves your un-committed modifications, in case you don't want to commit them yet, but need to switch branches or do anything that requires a clean working copy.
- When you write a commit message, the first line is known as the "subject line". It is a good practice to limit this subject line to 50 characters. In large scale project, commits should also match a given template. For example, it is a good idea to write your commits as follows :
  - the title should have the following pattern: `<file/module name>: <title>`
  - the commit should contain a brief description of what you have done in it.

  You can check the latest commits from one of Analog Devices GitHub projects in order to see examples: [9]. It is now straightforward that this kind of standard commit applies to single files; which is a much better practice than modifying tens of files at the same time.

4.3.5 Git homework

Goal: have all members of each group apply the workflow described section 4.3.3. You may create a temporary Git repository in Gitlab for practice if you wish, but at the end your work must be apparent in the history of your group's main project repository under `https://gitlab.telecom-paris.fr/projet-telecom/2020-2021`. Teachers *will* check. Don't forget to write meaningful commit messages!

You will find 2 short Matlab functions in your repository, folder `ref/git-homework`: `mandelbrot_calc` calculates whether given numbers belong to the Mandelbrot set; `mandelbrot_display` shows the Mandelbrot set as an image. Try them out, e.g.: `mandelbrot_display(-2, -1.25, 0.5, 1.25);`.

Over the course of the exercise, you will be asked to make modifications. For example, you could:

- uncomment the basic optimization in `mandelbrot_calc`;
- make the image size a `mandelbrot_display` function parameter instead of being hardcoded;
- make the number of iterations a `mandelbrot_display` function parameter instead of being hardcoded;

- display the image using a colormap instead of black-and-white, as per the comments at the end of `mandelbrot_display`...

Homework

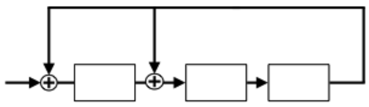
1. Coordinate so that each student successively makes a change and pushes it to the `master` branch before the next student pulls it and makes their change.
2. Do it again, working in parallel, each in your own temporary branch, as outlined in section 4.3.3. Designate one student to do the final merge into `master`.
3. Repeat as many times as necessary for every student in the group to have done a merge.

5 Appendix: BCH codes

5.1 Encoder for BCH codes

The encoding operations for generating BCH codes can be performed using linear feedback shift registers based on the generator polynomial $g(x)$. The main coding operation to handle is the polynomial modulo operation.

To illustrate the encoder, we take the example of the BCH code $(7, 4, 3)$ where the generator polynomial is $g(x) = 1+x+x^3$. The encoder calculates $m(x)\mathrm{mod}g(x)$. Let $m(x) = b_0x+b_1x^2+b_2x^3$



- Initially, the shift register contains all zeros.
- The coefficients of m(x) are clocked into the shift register one bit coefficient at a time, beginning by the highest coefficient, b_2, followed by b_1 and so on.

The contents of the shift register are as follows :

Inputs	Shift Register contents		
	0	0	0
b_2	b_2	0	0
b_1	b_1	b_2	0
b_0	b_0	b_1	b_2
0	b_2	$b_2 + b_0$	b_1

Consequently, $m(x)$ mod $g(x) = b_2 + (b_2 + b_0)x + b_1x^2$ are the contents of the shift register after 4 clock ticks.

5.2 Decoder for BCH codes

Use the decoding example for BCH codes from the TELECOM202B course. You have to create a matrix **M** which will contain the vector representation of the elements of $GF(32) = GF(2)[x]/1 + x^2 + x^5$ (last column of the following table). To build the matrix **M**, you can use the same modulo operation program used for the encoding process.

l	α^l	Polynomial representation	Binary representation
0	1	1	[10000]
1	α	x	[01000]
2	α^2	x^2	[00100]
3	α^3	x^3	[00010]
4	α^4	x^4	[00001]
5	α^5	$1+x^2$	[10100]
6	α^6	$x+x^3$	[01010]
7	α^7	x^2+x^4	[00101]
8	α^8	$1+x^2+x^3$	[10110]
9	α^9	$x+x^3+x^4$	[01011]
10	α^{10}	$1+x^4$	[10001]
11	α^{11}	$1+x+x^2$	[11100]
12	α^{12}	$x+x^2+x^3$	[01110]
13	α^{13}	$x^2+x^3+x^4$	[00111]
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Bibliography

[1] Analog Devices. *AD-FMCOMMS3-EBZ User Guide*. https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms3-ebz. 2020.

[2] Analog Devices. *ADALM-PLUTO for End Users*. https://wiki.analog.com/university/tools/pluto/users. 2021.

[3] Loren Shure. *Making Pretty Graphs*. MathWorks. Dec. 2017. URL: https://blogs.mathworks.com/loren/2007/12/11/making-pretty-graphs/.

[4] Greg Wilson et al. "Best practices for scientific computing". In: *PLoS biology* 12.1 (2014), e1001745. DOI: 10.1371/journal.pbio.1001745.

[5] Michael Robbins. *Good Matlab Programming Practices for the Non-Programmer*. 2001. URL: http://www.mit.edu/~pwb/cssm/GMPP.pdf.

[6] Rémi Sharrock. *TP SmartGIT*. URL: https://perso.telecom-paristech.fr/bellot/CoursJava/tps/tpgit.html.

[7] Jean-Claude Dufourd. *Transparents PACT*. URL: https://perso.telecom-paristech.fr/dufourd/cours/pact/pact-gl2n.html#/git-rappel-inf103.

[8] Serdar Yegulalp. *6 Git mistakes you will make — and how to fix them*. 2020. URL: https://www.infoworld.com/article/3512975/6-git-mistakes-you-will-make-and-how-to-fix-them.html.

[9] Analog Devices. *Analog devices LibIIO GitHub webpage*. https://github.com/analogdevicesinc/libiio/commits/master. 2021.