Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

An Open Source Project for Learning 5G Waveforms at an Electrical Engineering Graduate Level

Autor: Guillermo Palomino Lozano

Tutor: Juan Antonio Becerra González

Dep. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Máster

Máster en Ingeniería de Telecomunicación

An Open Source Project for Learning 5G Waveforms at an Electrical Engineering Graduate Level

Autor:

Guillermo Palomino Lozano

Tutor:

Juan Antonio Becerra González

Profesor Sustituto Interino

Dep. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Máster: An Open Source Project for Learning 5G Waveforms at an Electrical Engineering Graduate Level

Autor: Guillermo Palomino Lozano
Tutor: Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Acknowledgements

Con este documento termina mi formación educativa en un campo tan apasionante como es la Ingeniería de Telecomunicaciones. Atrás quedan 6 hermosos años, en los que he ido recorriendo un camino fascinante descubriendo y aprendiendo de este mundo. Y te vienen a la mente muchísimas personas que, en mayor o menor medida, te han ayudado o han formado parte de esta aventura.

En primer lugar, los profesores. Gracias a todos y cada uno de ellos, que con su esfuerzo consiguen que año tras año muchas personas salgan formadas como ingenieros, la profesión más bonita del mundo. Mención especial merece el tutor de este trabajo, Juan Antonio, que siempre ha estado ahí al pie del cañón. Siempre te estaré agradecido por todo tu empeño y dedicación a lo largo de toda mi formación académica. También me gustaría agradecer a la profesora María José Madero, que colaboró en la redacción del artículo asociado a este trabajo, y también por su extraordinaria docencia durante los últimos años.

En segundo lugar, los compañeros. Este proceso no solo te forma para poder ejercer tu profesión, también te forma para ser mejor persona. Y gran culpa de ello depende de la gente de la que te rodeas. Y, afortunadamente, me he sentido muy bien rodeado durante estos 6 años. Compañeros de clase convertidos en grandes amigos que se convertirán en compañeros de profesión. La unión hace la fuerza, creo que ha quedado más que demostrado. El *Team Polinino* nunca morirá.

En tercer lugar, merece mención especial este último año de mi vida, donde gracias a la beca Erasmus he podido vivir en el extranjero el mejor año de mi vida. Sin duda, la experiencia vivida allí, así como la formación recibida en una universidad extranjera, una vez más han servido para crecer aún más como futuro ingeniero y como persona. Gracias a todos y cada uno de vosotros, chicxs.

Por último, la familia. Esos que siempre están ahí y nunca se irán, que te agarran cuando te caes, que te echan una mano cuando todo parece desmoronarse. Gracias papás por educarnos tanto a mí como a mi hermana en unos valores que quizá no entendimos de pequeño, pero que sin embargo ahora sí que entendemos e intentaremos transmitir a las futuras generaciones: educación y respeto. Y gracias a ti, Sara, por entrar en mi mundo hace ya más de 7 años y hacerme la persona más feliz del mundo.

Mi etapa educativa finaliza en un momento de muchísima incertidumbre, en la que nuestras vidas se han visto alteradas en los últimos meses de forma drástica, marcadas por la pandemia COVID-19. Habrá cosas que nunca volverán a ser como antes, pero la ilusión y motivación permanecerán igual que el primer día.

THE END.

Guillermo Palomino Lozano
Alumno de 2º de MUIT

Entre Cáceres y Lund, 2020

Resumen

El presente documento describe la implementación de un software libre y de código abierto para la generación de formas de onda 5G-NR. El aprendizaje de las tecnologías de comunicaciones digitales requiere experiencia, y los estudiantes a menudo exigen contenidos más prácticos para entender los conceptos básicos de dichas asignaturas. Paralelamente, el desarrollo comercial de la quinta generación de comunicaciones móviles (5G) comenzó en 2020. 5G pretende ser una tecnología inalámbrica revolucionaria en términos de latencia, velocidad de datos, eficiencia energética y comunicación masiva. Los estudiantes deberán saber cómo funciona la capa física de 5G, conocida como 5G New Radio (5G-NR) y, por lo tanto, deben hacer frente a los desafíos de diseño del sistema que existen en esta tecnología. La herramienta presentada está contextualizada en el marco de las Ingenierías de Telecomunicación. Además, el software presentado aquí también será útil para los investigadores que podrán utilizar las formas de onda de 5G-NR para sus propios proyectos de investigación.

Abstract

This paper describes the implementation of a free and open-source software for 5G-NR waveform generation. Learning digital communication technologies skills requires experience and the students often demand more practical contents in order to understand the basic concepts of their courses. Linked to this, the commercial development of the fifth generation wireless system (5G) started in 2020. 5G aims to be a revolutionary wireless technology in terms of latency, data rates, energy efficiency and massive communication. Digital Communication students must know how 5G works and, therefore, they must study its physical layer and the system design challenges that exist in this technology. The presented tool is contextualized in Engineering curricula, more precisely in a program named Electrical and Computer engineering. This tool developed will also be helpful to researchers that will be able to use 5G-NR waveforms for their projects.

Summarized contents

Contents

Abbreviations and Acronyms

$3GPP$	Third-Generation Partnership Project
$5G$	Fifth Generation wireless system
$AWGN$	Aditive White Gaussian Noise
BER	Bit Error Rate
$CDMA$	Code-division Multiple Access
DPD	Digital Pre-Distorsion
$eMBB$	enhanced Mobile Broadband
FDD	Frequency Division Duplex
$FOSS$	Free and Open-Source Software
FR	Frequency Range
GSM	Global System for Mobile Communication
ICI	Intercarrier Interference
ISI	Intersymbol Interference
IoT	Internet of Things
$IMT-2020$	International Mobile Telecommunications 2020
$ITU-R$	International Telecommunications Union - Radio communications sector
LTE	Long Term Evolution
$mMTC$	massive Machine Type Communications
$M2M$	Machine-To-Machine
$NMSE$	Normalised Mean Squared Error
NR	New Radio
$OFDM$	Orthogonal Frequency Division Multiplexing
$PAPR$	Peak to Average Power Ratio
$PyPI$	Python Package Index
RB	Resource Block
RE	Resource Element
TDD	Time-Division Duplex
$UMTS$	Universal Mobile Telecommunications System
$URLLC$	Ultra-Reliable and Low-Latency Communications

1 Introduction

Why should I try to make you believe the things I believe in?

Dennis Rodman, a controversial NBA basketball player

Mobile communications have redefined the relationship between human beings in the last 40 years in terms of society and technology. Everything started in 1973 when Martin Cooper (a Motorola executive) made the first phone call ever. At that moment a race for bringing this new technology to the user market started, and it meant that around 1980 the first generation of mobile communication emerged. This very first system only allowed voice serviced but, for the first time in history, allowed ordinary people to make phone calls to every part of the world.

The second generation of mobile communication (2G) came out in the first years of the 1990s. But there was a lack of a unified system for serving calls all over the world. One of the most famous systems was the Global System for Mobile Communication (GSM) that was developed by some European countries. GSM dominated the implementation of 2G because it spread to other non-European countries and it is considered the first global system for mobile communication. 2G was mainly designed for serving voice calls but it included digital transmission to provided limited data services.

People started to use a cell phone in their daily lives so it was necessary to introduce a new generation. In early 2000 the third generation of mobile communication (3G) was born, with the main difference that now it uses packet switching instead of circuit switching for data transmission. This technology was known as High-Speed Packet Access (HSPA) [GMB, 2006]. Additionally, a new core network architecture was completely designed, called Universal Mobile Telecommunications System (UMTS), and the channel access technique used was Code-Division Multiple Access (CDMA). Therefore, a big step of quality was taken and users observed how data rates increased. A new era had just begun, and users started to connect to the Internet with their cellphones.

The fourth generation (4G) was deployed in 2013 and went one step further providing not only higher data rates but also more devices connected. 4G is a technology that understands that the number of devices in the society is growing exponentially and therefore the services that those devices can offer are more than just calls and messages. Cellphones are now smartphones, and they demand a stable Internet connection for video streaming, social media, augmented reality, etc. This generation involves the Long Term Evolution (LTE) technology and the channel access is Orthogonal Frequency Division Multiplexing (OFDM).

Almost half a century after Martin Cooper made the first call, the fifth generation (5G) wireless system deployment is starting. A summarized evolution of the different generations of mobile communications is shown in Fig. 1.1. But 5G is not just an upgrade of last generations of mobile communication [Gupta and Jha, 2015, Ghosh et al., 2019]. Far from being only an evolution of the latest generation wireless technology, 5G is a system with new services capabilities that expect to meet the demands imposed by the users in the coming years. According to [Cis, 2020], by 2023 a 66% of the population will have Internet access —up from 51% in 2018— and the 29.3 billion devices connected to this global network will represent more than thrice the global population. Additionally, there will be an increase in the number of devices in machine-to-machine (M2M) communication, growing from 33% of the total devices in 2018 to 50% by

2023. The increment in data rates and the number of connected devices to legacy mobile generations pose a challenge to standards that are fulfilled with 5G [Ijaz et al., 2016, Henry et al., 2020].

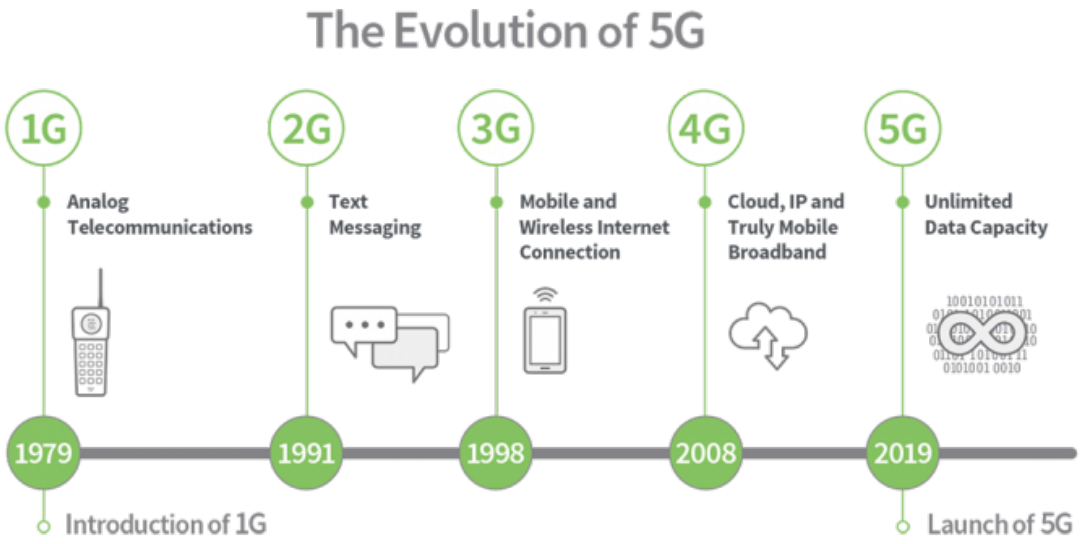

Figure 1.1 Road to 5G [Remmert, 2019].

After initial 5G launches in 2019, it is expected that commercial telecommunication companies start to offer 5G connection in most of their territory in the following years. The key for a successful technology is in its process of standardization that must allow interoperability between infrastructure and devices from different companies. The specifications of the new standard have been created by the Third-Generation Partnership Project (3GPP), as they did with previous cellular technologies such as Universal Mobile Telecommunications System (UMTS) and Long-Term Evolution (LTE) [ROA, 2020]. 3GPP is following the International Telecommunications Union - Radio communications sector (ITU-R) guidelines, submitting a candidate technology that will be evaluated following the International Mobile Telecommunications 2020 (IMT-2020) requirements [Marcus, 2015]. Taking into account the necessities of the society today, IMT-2020 specified three use case scenarios [ITU-R, 2015]:

- *Enhanced mobile broadband (eMBB)*: Requirement of high data rates across a wide coverage area. Since the end-user data-rate demand is continuously increasing, higher frequency deployment is particularly suitable for dedicating more bandwidth for the transmission.
- *Ultra-reliable and low-latency communications (URLLC)*: This case is considered for scenarios where latency requirement is crucial. Additionally, very high availability and reliability must be guaranteed. Examples of scenarios that cover this situation are those related to autonomous driving and automatic control.
- *Massive machine-type communications (mMTC)*: services that require a large number of entities connected simultaneously. These low-cost devices such as sensors or actuators consume ultra-low power. Their requirements also include ultra-low latency, high reliability, and availability.

1.1 Scope and challenges

This work presents a free and open-source software (FOSS) platform for learning 5G New Radio (5G NR), the radio access technology of 5G, that is intended for its use at undergraduate and graduate-level courses. Previous studies have presented FOSS frameworks for 5G deployment, but focusing on other aspects, such as a techno-economic assessment [Gomes et al., 2018] or communication between networks

[Lin et al., 2017, Kaltenberger et al., 2019, Salama and Elmesalawy, 2019]. This software is developed in the Python programming language and it is publicly available within SciPy, the Python-based ecosystem of open-source software that is a common tool for laboratory activities at engineering programs, and Codeocean, a research collaboration platform by IEEE.

The objectives of this work can be enumerated as follow:

1. Study the origin of the mobile communication systems and how they have progressed during the last decades, highlighting in the most recent technologies (LTE and 5G NR).
2. Try to know better about how 5G-NR waveforms are generated and how to choose the parameters needed for the transmission efficiency.
3. Develop a FOSS platform for learning 5G-NR waveforms using Python programming language.
4. Learn which are the most important figures of merit in 5G NR and try to understand which values are considered as admissible parameters in this technology.
5. Provide an on-line tool for future students that helps them to understand how 5G NR works.
6. Bring a closed-box package that generates 5G-NR signals that researchers and users, in general, can use to develop their simulations and experiments.

1.2 Planning

Since this works consists of a real project that is going to be implemented, it was considered to create a Gantt chart to schedule and define all the tasks that had to be performed. The fact of creating this chart was helpful because it allowed me to separate tasks, and it was also helpful to the supervisor, that knew in every moment how was the project status. The Gantt chart is represented in Fig. 1.2. It is possible to identify two different groups of tasks. One called *software development* and another one called *documentation and publication* then all the documentation and publication process. All tasks are explained in detail next:

- **Software development**.
  - **Problem definition**. This part consisted of some meetings where the project supervisor explained to the student the problem they had and the reasons why they wanted to develop a platform like this.
  - **5G-NR readings**. It is true that the student had some previous knowledge in 5G NR but it was necessary to read some books to get background knowledge of how exactly 5G-NR standardization process is doing. Once all this knowledge was gotten, I was ready to start coding.
  - **Toolbox implementation**. This task is the main one of this part, where all implementation is done taking more than two months. This task was coordinated with the next one, which is the verification and simulation process. As the reader can think, the verification process will return modifications in the toolbox: that is the reason why both share space in time.
  - **Verification and simulations**. Once most of the tool is developed, it is convenient to start with the simulations that will be explained in the following chapters. Additionally, some tests were performed to verify the system developed.
- **Documentation and publication**.
  - **IEEE Access paper preparation**. It was convenient to write a paper explaining the tool performed and the impact it will have on Electrical Engineering students. Therefore, a task was defined, whose objectives were to start explaining and defining the software.
  - **Pypi package preparation**. Additionally, the tool was uploaded to the Pypi package. It needs some modifications regarding the structure and some files that must be defined.
  - **Codeocean capsule preparation**. Doing the same way, the purpose of this task is to adapt the tool to the Codeocean format.

– **Thesis elaboration**. Finally, this project finishes with the thesis elaboration, that is this document that you are reading. The thesis elaboration includes a defense with a presentation that must be also prepared in these tasks.

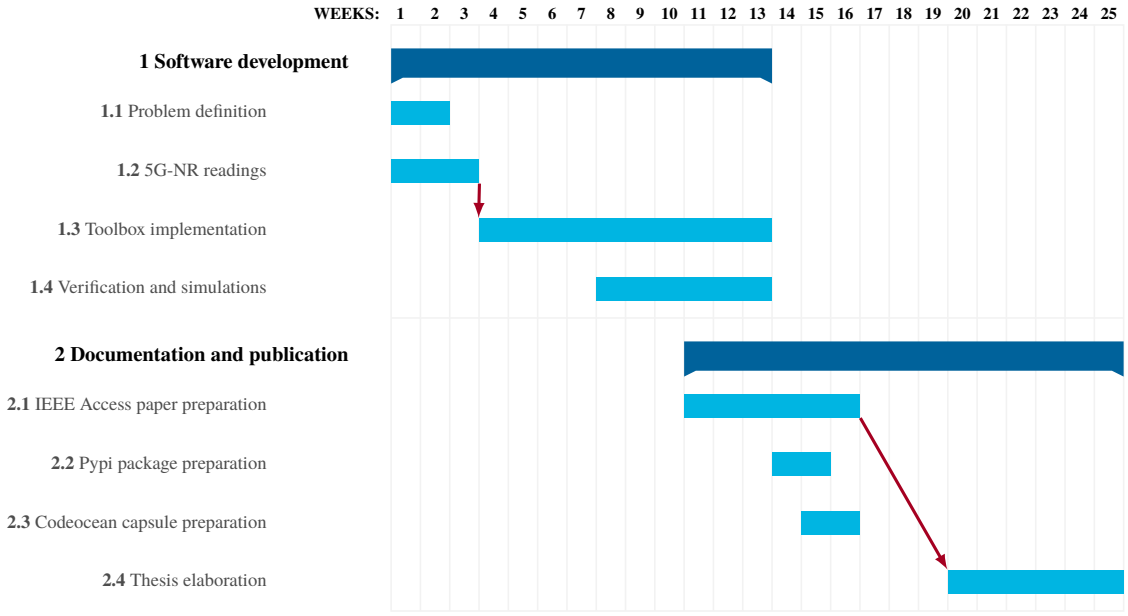

Figure 1.2 Gantt diagram associated to the project planning.

1.3 Structure of this work

This thesis is organized into four chapters, where the first one is the one you are reading now, and it is an introduction for the rest of the work. As you can understand, this first chapter is useful for the reader in terms of motivation and understanding. The rest of this is organized in three chapters, very different from each other:

- **Chapter 2: 5G New Radio overview**. This second chapter presents an overview of 5G-NR technology on which this project is based. We will motivate the use of numerology, key for considerate several usage scenarios. A brief recap of how OFDM works is also included here, and how 5G-NR adapt OFDM for their transmissions. Signal generation, both frequency and time domain is also considered here. This chapter finishes with some necessary definitions that everyone who works with 5G NR should be familiar with.

- **Chapter 3: design of the proposed tool**. The complete description of the designed tool is introduced in this chapter. The design is organized in seven different blocks: random data generator, modulator, oversampling, Inverse Fast-Fourier Transform (IFFT), Cyclic Prefix, Peak clipping, and Spectrum shaping filter. Every block is explained in detail in a mathematical way, related at the same time with the code that has been developed. This chapter is the key to the whole thesis and it is a basic one that the reader must understand before reading the next one, that is where all simulations and results are presented.

- **Chapter 4: results and discussions**. Once the tool has been completely explained, the next step is to present all the simulations that have been carried out and the results that are extracted after that. The tool presented here allows a lot of functionalities, therefore a variety of different simulations may be performed. The motivation for creating new simulations is also explained here. This chapter ends with some future lines that could continue the work done here and also some conclusions.

1.4 Software employed

For the platform implementation, the chosen programming language was Python. The reasons why Python was selected are explained next, but it is also important to know that the author has already some experience with this programming language. Hence, most of the work is to focus on developing the tool itself and not learning the programming language. However, there is always something new that must be learned, like new libraries or some code techniques that the developer must understand and therefore spend some time on learning those tasks. In any way, some of the Python features that are helpful for this particular project are [Oliphant, 2007, Fangohr, 2004, Ozgur et al., 2017]:

- It is a FOSS, so there is an active community of developers continuously creating projects that users can employ for their activities. Hence, it helps to reduce development costs significantly. It is commonly known that Python has strengthened the past years. If you go search in Google which are the most famous programming languages Python will be for sure the top 3 in each of them.
- There are a large available number of libraries that help to optimize the code. Some of the most important libraries and packages for signal processing used in this project are: *Numpy*, that provides support for matrices and vectors enabling efficient implementation in a high-level language [van der Walt et al., 2011]; *Matplotlib*, a data visualization package with a graphical interface similar to the Matlab programming language that provides high-quality output in many formats, including portable document format (PDF), encapsulated postscript (EPS), and portable network graphics (PNG) [Ari and Ustazhanov, 2014]; and *SciPy*, more for a general purpose on scientific and technical computing [Oliphant, 2004]. The latter is part of the *Numpy* project.
- The Python Package Index (*PyPi*) is the official software repository for Python, where hundreds of thousands of developers upload their packages. Users can easily download, install a package, and add them to their projects. Precisely, this project is uploaded to the *PyPi*, so it is easy to install and download for anyone.
- Users can run external applications with Python, as well as embed Python into another one. Consequently, the abstraction level is increased. As an example, in this project, an application developed in Matlab located in an external server is run.

To put it briefly, Python is one of the most important programming languages in the world of today, and it also applies in mobile communication where every day more and more projects are developed in Python because of its robustness.

2 5G New Radio Overview

Good ideas are always crazy until they are not.

ELON MUSK, BILLIONAIRE CEO OF SPACEX AND TESLA.

2.1 Motivation

The purpose of this second chapter is to give a 5G background before start commenting on how development tool has been designed. Our starting point is the fourth generation of mobile communication, LTE. One reason for doing that is that both are developed by the same organization, 3GPP. Hence, 5G NR can be considered as an upgrade of LTE in terms of radio access, providing the following benefits:

- Lower latency, that allows new scenarios.
- It is possible to exploit higher frequencies, which is directly related to the use of more spectrum for getting more data rates.
- Researchers have focused on what is known as ultra-lean design. It improves network energy efficiency, managing the transmissions in a way that the "always-on" transmissions are minimized, according to an Ericsson report [Zaidi, 2017]. This technique is without any doubt an advantage because of two reasons: first, it is better for a sustainable society and second, it reduces operational expenses.
- 5G NR is prepared for forwarding compatibility, for future use cases and technologies.
- If we have to describe 5G-NR technology with one word, it is clear: flexibility. It will be mentioned during all the chapters but 5G NR must address all of the different use cases already mentioned in the last chapter (eMBB, mMTC, URLLC), so a range of carrier frequencies, order modulation and, in general, different transmission schemes must be defined dynamically. The software that is presented in this project covers most of these aspects, see next chapter for further details.

2.2 OFDM as 5G-NR waveform

Following the same procedure which was done with the development of LTE, orthogonal frequency division multiplexing (OFDM) was chosen as the waveform for 5G-NR transmissions [Guan et al., 2017]. OFDM is primarily chosen due to its spectrum efficiency, relative immunity to selective fading, and simple channel equalization. Additionally, it is also easy to exploit both the time and frequency domains. OFDM allows dividing the available bandwidth into parallel subchannels that are called subcarriers. Some of the advantages of using OFDM include:

- No equalization of delay spread is needed. As a consequence, the OFDM received symbols does not experience any ISI.

Table 2.1 Numerology in 5G NR.

μ	Δf [kHz]	T_u [μs]	T_{CP} [μs] for symbols 1 to 6
0	15	66.7	4.7
1	30	33.3	2.3
2	60	16.7	1.2
3	120	8.33	0.59
4	240	4.17	0.29

- It is a modulation that is robust to selective fading. The channel is divided into orthogonal signals: a selective fading is therefore converted into a flat fading for one subcarrier.

- High spectral efficiency. The subcarriers are close to each other which is good for spectral efficiency.

But OFDM has also some disadvantages, such as high Peak to Average Power Ratio (PAPR), which is defined as the ratio between the peak power and the average power of a signal, commonly expressed in decibels. As a consequence of a high PAPR, there will be a high dynamic range, therefore a highly linear power amplification will be needed. Amplifiers with a high dynamic range usually are power inefficient. Another technique commonly used is to reduce PAPR using peak clipping techniques. This one is the one that is implemented in this project. Another disadvantage of OFDM is that it is sensitive to carrier frequency offset, that arises a phenomenon known as Intercarrier Interference (ICI), that deteriorates the quality of the transmission.

The physical layer designed in 5G NR allows OFDM for both the downlink and uplink. Additionally, discrete Fourier transform (DFT) precoded OFDM can be used as a waveform in the uplink due to its efficiency in energy-limited scenarios. According to [Truong et al., 2014], the energy consumption of the power amplifier may be reduced by a factor of two. But the use of DFT-precoded OFDM also has some drawbacks. To give an example, spatial multiplexing becomes more complex, and also using this scheme the possibility of allocating resources in non-contiguous frequencies is lost, so it implies scheduling restrictions as well as losing frequency diversity.

2.3 Numerology

In a 5G-NR context, the term *numerology* is associated to the design parameters that can be defined for a specific waveform. In LTE, there is only one type of subcarrier spacing of 15 kHz. In 5G this is completely different since one of the main challenges is to support different use case scenarios [Osseiran et al., 2014] therefore it is not possible to have just one numerology. A scalable numerology is needed. As a result, a completely new one is defined and it is shown in Table 2.1. To give an explanation of how numerology works in 5G NR, it is convenient to discuss regarding the definition of Discrete Fourier Transform operator (DFT).

2.3.1 Discrete Fourier Transform

Waveform generation in 5G NR is based on OFDM, which is straightforward related to the DFT, defined as

$$X(k) = \frac{1}{N} \sum_{k=0}^{N-1} x(n) e^{-j2\pi kn/N}, \tag{2.1}$$

where $x(n)$ is the signal represented in a sequence of N complex number in discrete-time domain, and $X(k)$ the output of the DFT operation, also a sequence of complex numbers. It is also possible to define the inverse operation, the Inverse Discrete Fourier Transform (IDFT),

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) e^{j2\pi kn/N}. \tag{2.2}$$

The most common algorithm that computes the DFT and the IDFT is called the fast Fourier Transform (FFT) and the inverse Fourier Transform (IFFT) respectively, and they are widely used in 5G-NR waveform generation. Each subcarrier is modulated depending on one transmitted symbol, i.e., the modulation is performed in parallel, as it is shown on Fig. 3.3. It means that every T_u seconds an IFFT with FFT_{size} points is performed. The sampling frequency F_s is defined as follows,

$$F_s = \text{FFT}_{\text{size}} \cdot \Delta f, \tag{2.3}$$

where FFT_{size} is the FFT size, that must be power of two in order to be computationally efficient, and Δf is the subcarrier spacing, whose expression is

$$\Delta f = 2^{\mu} \cdot 15, \tag{2.4}$$

where $\mu \in [0,4]$ is the parameter that defines the numerology. Available configurations are shown in Table 2.1.

2.3.2 Cyclic Prefix

After the IFFT operation, the cyclic prefix (CP) is added. In LTE, two types of CP are specified: normal CP (NCP) and extended CP (ECP). In practice, ECP was not used, so 5G only defines a NCP except for $\Delta f = 60$ kHz, situation in which both types of CP may be used, so that a CP similar to the $\Delta f = 15$ KHz case can be maintained. Following the 3GPP standard [ETS, 2018], the length of the CP, in samples, is defined as follows,

$$N_{\text{CP}}^{\mu} = \begin{cases} 512\kappa \cdot 2^{-\mu} & \text{ECP} \\ 144\kappa \cdot 2^{-\mu} + 16\kappa & \text{NCP for symbol 0} \\ 144\kappa \cdot 2^{-\mu} & \text{NCP for symbols 1 to 6} \end{cases} \tag{2.5}$$

where κ is a constant that relates LTE and 5G basic unit times and takes a value of $\kappa = 64$. Note that the first symbol of every 7 symbols exhibits an extended duration in time. The CP duration follows

$$T_{\text{CP}} = N_{\text{CP}} \cdot T_c, \tag{2.6}$$

where T_c is the basic 5G time unit that is defined as $T_c = 1/(480000 \cdot 4096)$.

Since the program allows to generate a signal with a custom sampling frequency of F_s, it is necessary to adjust the number of samples accordingly,

$$N'_{\text{CP}} = N_{\text{CP}} \cdot \frac{T_u}{T_s}, \tag{2.7}$$

where N'_{CP} is the new length of the CP after the adjustment. Once the numerology has been defined, the next step is to create the time structure of the signal. In 5G the transmission schemes are defined in two domains: time domain and frequency domain. Both are reviewed next.

2.4 Duplex schemes

One of the keys of the wireless systems is how to maintain the communication link in both directions. Spectrum flexibility is one of the features of NR. It ensures separation of uplink and downlink in frequency and/or time domains. Duplex schemes defined in NR are Time-Division Duplex (TDD), Frequency-Division Duplex (FDD), and Half-Duplex FDD. All of them are briefly explained next.

2.4.1 Time-Division Duplex (TDD)

It uses only a single frequency for transmission and reception. Hence, there is not exist and overlapping between uplink and downlink transmissions. What is more, NR uses what is called *dynamic TDD*

[Dahlman et al., 2018]. *Dynamic TDD* ensures the system to dynamically allocate resources for uplink or downlink depending on the demand in a certain moment. It means that it is easy to adjust the capacity in either direction. It is useful in small-cell and isolated cell scenarios. However, for large distances it is not a good technique: a guard period is added to solves the propagation time issues.

2.4.2 Frequency-Division Duplex (FDD)

It requires one channel for uplink transmissions and another for downlink transmissions. Nevertheless, both links can be transmitted at the same time. If the capacity must be adjusted, it is possible to re-allocate channels. This technique solves the problem of the period guard but the complexity is higher (duplex-filters must be added). Therefore, the costs increase. That is the reason why another duplex scheme is introduced.

2.4.3 Half-Duplex FDD

In this case, the device uses different frequencies for uplink and downlink and they do not transmit at the same time. This is a simplified technique that reduces device costs, for cases where there are no duplex-filters. Since the devices cannot transmit in all uplink/downlink subframes, the data rate is reduced in this case.

2.5 Time domain

5G transmissions in time domain are structured into *frames* of 10 ms. One frame is, in turn, divided into 10 *subframes* of 1 ms each. A subframe is again divided into several slots that contain 14 OFDM symbols each. The number of slots allocated in a subframe depends on the numerology. An example of how slots are organized in 5G is shown in Fig. 2.1, in which shaded symbols denote a symbol whose CP is longer. It is possible to see in the figure that the structure is scaled dividing by a power of two, with the origin in 15 KHz for coexistence with LTE. Another benefit of this division is that mixing different numerologies in the same subcarrier is avoided.

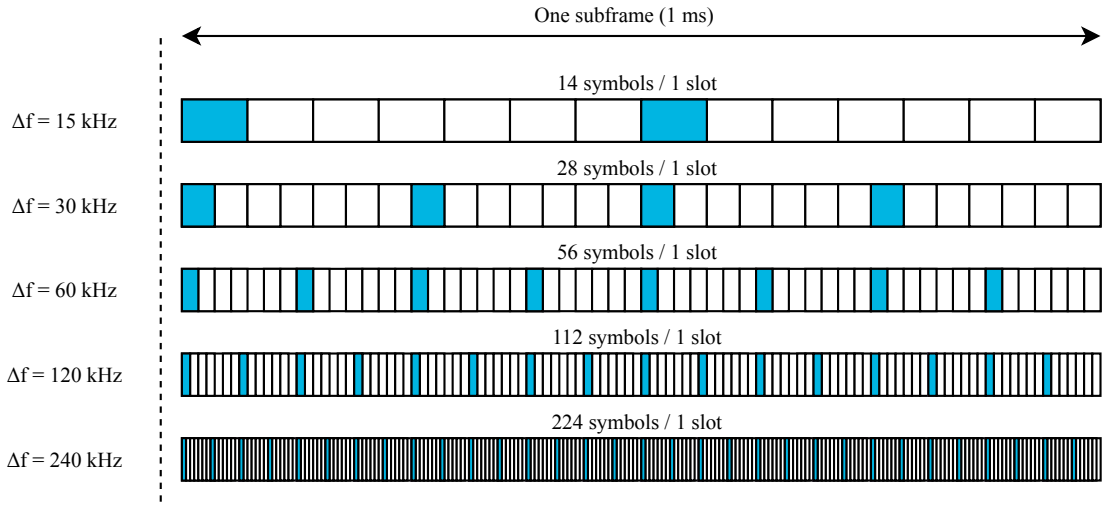

Figure 2.1 Slots in 5G according to subcarrier spacings of 15, 30, 600, 120, and 240 kHz.

It is also important to comment that there is a technique implemented in 5G NR to support low latency transmissions. It consists on separate the concept of transmission duration from slot duration. In other words, it is possible to use only a portion of the slot for transmission. It is known as *mini-slot* transmission.

Table 2.2 Frequency range related to numerology in 5G.

Frequency range	Δf (kHz)	Number of RB	BW_{ch} (MHz)
FR1	15	25-270	5, 10, 15, 20, 25,
	30	11-273	30, 40, 50, 60, 70,
	60	11-135	80, 90, 100
FR2	60	66-264	50, 100, 200, 400
	120	32-264	

2.6 Frequency domain

The basic element in terms of frequency is a Resource Element (RE). It consists of one subcarrier during one OFDM symbol. Another division commonly used is referred to as a Resource Block (RB), which consists of twelve consecutive subcarriers. It is important to remark here the difference of a RB in LTE, which was twelve subcarriers during one OFDM symbol **and one slot**. Here an RB is a one-dimensional measure, and it can be explained if we understand that 5G NR is more flexible. As was commented in the last section, now the transmission does not have to occupy the entire slot, and that is the reason why the definition has changed.

3GPP introduced in Release 15 two ranges of frequency where a 5G transmission can be allocated, which are called frequency ranges (FRs). Sub 6 GHz range is called FR1 and the millimeter-wave range is called FR2. FR1 covers bands from 450 MHz to 7.125 GHz and FR2 from 24.25 GHz to 52.60 GHz. [1] The selection of an FR involves a different utilization of the spectrum. Note that not every numerology is allowed in each FR. Table 2.2 shows the number of RBs allowed for each situation as well as the channel bandwidth specified for each case.

[1] For the full list of the bands involved in the different frequency ranges, see *https://www.cablefree.net/wirelesstechnology/4glte/5g-frequency-bands-lte/*

3 Design of the proposed tool

Your love makes me strong. Your hate makes me unstoppable.

CRISTIANO RONALDO, FOOTBALL PLAYER.

After introducing all 5G-NR concepts that are necessary for the complete understanding of this work, next step is to introduce how the development tool looks like and also how it has been designed and implemented. This chapter is organized into two parts. The first one, where the tool itself is presented and we will discuss everything based on the flowchart and the block diagram of the tool. The second part of this chapter consists of a description of the repositories used to upload this project.

3.1 Software designed

3.1.1 Flowchart description

To start with, the flowchart of the full project is shown and can be found in Fig. 3.1, and the full code of the developed tool is included in the appendix (Code A.2 and Code A.1). It consists of a full simulation of 5G generation and their following reception. As it has been already commented during the first chapter, the tool has been developed in Python, which is an open-source programming language.

The software starts with some configuration parameters, that can be set by the user but they must be according to the standard defined, that has been commented in the last chapter (table 2.1 and 2.2). More precisely, the parameters that the user can set are:

- **Number of Resource Blocks** (`NRB`). Here the number of RB is set and must be defined according to Table 2.2.
- **Subcarrier spacing** (`Df`). Subcarrier spacing is another parameter that must be set in order to prepare everything for the transmission.
- **Number of slots** (`Nslots`). It indicates the number of slots that are going to be generated. The more slots that are generated, the more accurate our calculates will be, but it also will take more resources on our computer, and more time will be spent to calculate the results.
- **Frequency range** (`FR`). As has been commented, 5G-NR transmission can be allocated in two frequency ranges. By setting FR1 or FR2 we fix the numerology we are using, that is completely different.
- **Modulation order** (`M`). It represents the size of the constellation used for the transmission. One of the following schemes supported must be chosen [ETS, 2018]: binary phase-shift keying (BPSK), quadrature phase-shift keying (QPSK), and quadrature-amplitude modulation (QAM) of 16QAM, 64QAM, and 256QAM.

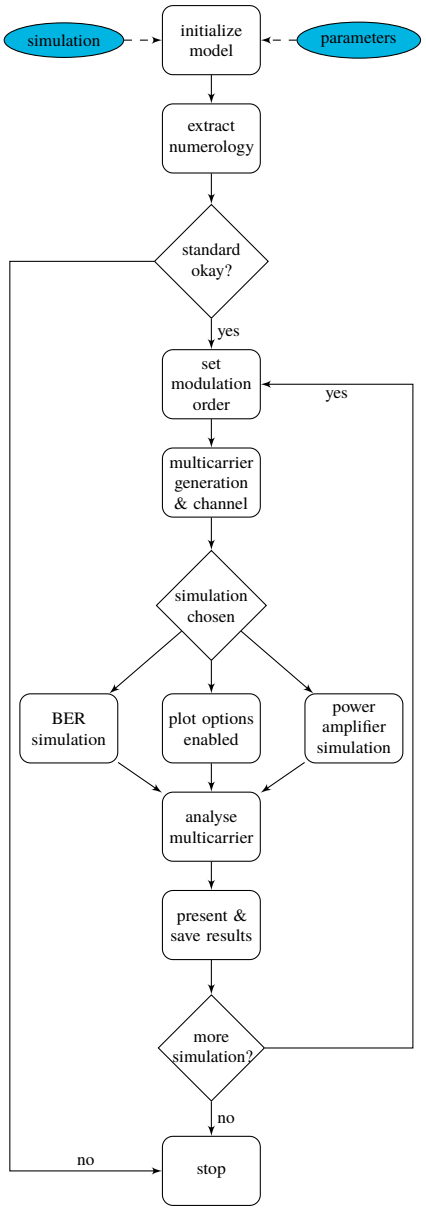

Figure 3.1 Flow chart of the simulation script.

- **Peak to Average Power Ratio** (`PAPR`). We were discussing in the last chapter that PAPR was one of the drawbacks of using OFDM as the modulation technique. This parameter set what is the maximum PAPR value admissible. If it is higher, it will be reduced by peak clipping techniques.

- **Seed for random number generation** (`SEED`). This parameter intends to set the seed for random number generation, commonly used for developers.

Additionally, there are also some boolean variables that the user can change, and that allows us to run one specific simulation or just avoid the representation of figures to be more computationally efficient. Regarding those parameters, they will be commented in detail in the next chapter where all simulations that have been carried out are presented.

As was said a few lines ago, before starting the simulation, the tool checks that every parameter is defined according to the standard. If that is not true, the simulation will stop, showing in the screen that the parameters set are not defined according to the standard, and it shows how the format should be. An example is shown in Fig. 3.2, where a certain number of RB of 1 is set, but the program returns a message specifying the range of the RB that can be set for the given parameters.

```
------------- STARTING TX -------------

 M = 64
 From Fo = 0.48 MHz to Fs = 0.48 MHz
The number of resource blocks does not match the defined numerology

Desired range: 11-273

Your value: 1


In [4]:
```

Figure 3.2 Screenshot that shows and example of what happen when parameters specified does not match the standard of 5G NR.

Once the parameters have been defined properly, the next step is to specify the modulation order. It is done like this because there is a simulation that uses all the modulations to calculate some results and compare them for every modulation. Hence, if it is defined here and not before, it is possible to repeat the simulation only varying that parameter, and they can be allocated under a for a loop. After that, the set-up would be complete, and the next step is to generate the 5G-NR signal and after that, receive it. All this process is described in detail in the next section, where a block diagram that represents this action is commented. Lastly, all results must be saved but additionally, some of then are printed and can be seen in the console log. If more simulations are necessary to do, the software will be run again, otherwise, it will finish and we will be able to see all the figures that have been generated.

3.1.2 Block diagram

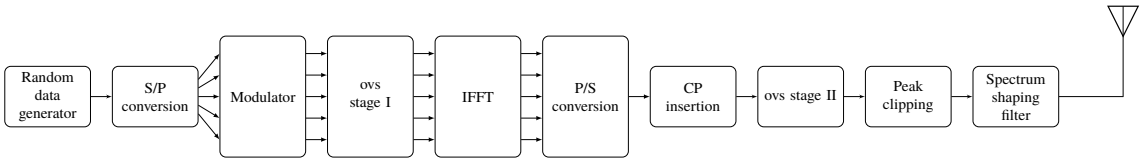

Figure 3.3 Diagram of the multicarrier generation stage (ovs stands for oversampling).

The block diagram of the transmitter stage is shown in Fig. 3.3. In this section, waveform generation is discussed. For waveform reception, the steps defined are the same as in the generation stage, but backward. For simplicity, we will only comment on the transmitter stage. There are some stages with more than one arrow, which means that the information in those stages is processed in parallel. Each step is explained next.

Random data generator

First of all, a random sequence of bits must be generated. Therefore, it is necessary to calculate how many bits are needed. It is important to remark that the bit sequence will be determined by the seed value already commented in the last section. Taking into account the number of slots and resource blocks that have been specified, it is possible to calculate the number of bits that must be generated for the transmission,

$$N_{\text{bits}} = (14\, N_{\text{slots}}) \cdot (12\, N_{\text{RB}}) \cdot \log_2(k), \tag{3.1}$$

where N_{slots}, N_{RB} and k are the number of slots, RBs and the order modulation, respectively. This equation comes from the fact that we need to insert k bits into an specific subcarrier every symbol. $12\, N_{\text{RB}}$ is the number of subcarrier per OFDM symbol and $14\, N_{\text{slots}}$ the number of OFDM symbols specified.

To give some understanding of the equation defined above, it is important to remark that the number of bits generated is influenced mainly by two variables: the number of slots and the number of resource blocks. We will see in the next chapter that if we want to run a simulation we must generate a significant number of bits, so it is important to take into account that there are two ways of doing that: time domain (slots) and

frequency domain (resource blocks). The first one is unlimited since we can always generate more symbols and therefore exploit the time domain. However, the number of RBs is limited at some point by the standard.

Modulator

Once that bits have been generated, the next step is to modulate them. For this purpose, it is necessary to obtain the level of amplitude for each complex symbol in both in-phase (I) and quadrature (Q) components. First, we get a constant that guarantees the desired energy per bit E_b in Joules,

$$A = \sqrt{\frac{3E_b \log_2(M_1 M_2)}{M_1^2 + M_2^2 - 2}}, \tag{3.2}$$

where M_1 and M_2 are the order modulation for I and Q components, respectively. In 5G NR, all modulation schemes are squared, so $M_1 = M_2 = \sqrt{M} = 2^k$. The alphabet that represents every level of amplitude, A_I and A_Q is obtained as follows [Proakis and Salehi, 2008],

$$\begin{aligned} A_I &= A(2i - M_1 + 1) \\ A_Q &= A(2j - M_2 + 1), \end{aligned} \tag{3.3}$$

where i and $j = 0,1,...,M-1$. Then, a bit sequence must be associated with each symbol. The best way to do that is by using a Gray codification, that ensures the minimum BER for a given symbol error rate (SER). After that, the bit sequence has turned into a digital signal that contains the bits mapped into symbols and can be represented into a constellation To show an example, Fig. 3.4 represents the constellation of every symbol for a given modulation order. The energy per bit value has been set to 1 during all experiments.

Once the symbol has been calculated, the next step is to associate them with the subcarriers. Thanks to the simplicity of OFDM modulation, this association is a direct operation because the value of each symbol corresponds to the value of each subcarrier. It is important to remark that the symbol generated is represented in the complex plane, as can be seen in Fig. 3.4. As a consequence, the subcarrier is also complex and if we want to plot the subcarriers, we will have to represent the absolute value of them instead of the real or the imaginary part. This is a concept that engineers that work with signals know very well.

Oversampling

To meet the requirements for other external applications, sometimes it is necessary to apply an oversampling (ovs) to increase the sampling rate. The natural way of doing that is at the final stage, where an interpolation may be performed to add/remove samples and change the sampling rate. However, this method distorts the spectrum of the signal and therefore it is necessary to explore other techniques to change the signal sampling frequency. One technique is to perform part of the oversampling in this stage. After modulating the subcarriers individually, it is possible to add zeros to the left and the right of the subcarriers. As a consequence, the number of samples will increase, but since we are adding zeros on both sides, the information stored will not change.

It is important to remark that only an equal number of zeros must be added on both sides. That means that only an oversampling with an integer value can be performed here. It follows that if an oversampling with decimals is required, an additional fine oversampling to the final rate will be performed before the clipping stage. But, at least, most of it will be performed here and, hence, the distortion is minimized.

The second part of the oversampling is performed at the end of the transmitter stage, after the CP insertion. However, it is going to be explained here because we have already put into context this concept. As it was said at the beginning of this section, if it is possible, the oversampling will be performed in the frequency domain, adding zeros to the left and the right of the subcarriers if it is an upsampling, or removing zeroes if it is downsampling. But sometimes the overall oversampling that must be performed will not be an integer number and therefore we will need to explore other techniques to change the frequency sampling accordingly. What has been done here is a technique that is called FFT interpolation [Selva, 2015].

Lastly, it must be clear that the purpose of the oversampling is to change the frequency sampling and an oversampling technique that works in the frequency domain will be always preferable: the distortion of doing a resampling is minimized if we face the problem in this domain.

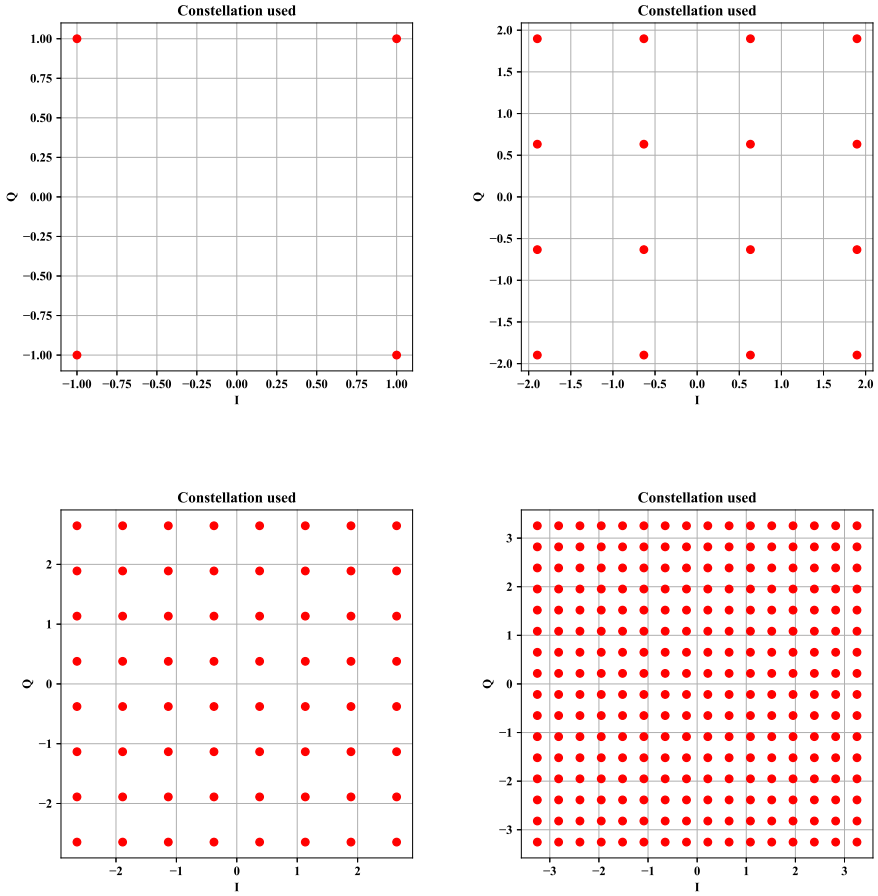

Figure 3.4 Constellation for all the modulation order generated in 5G NR: QPSK, 16QAM, 64QAM and 256QAM.

IFFT

The next step is to perform the IFFT. It has also been considered that more than one OFDM symbol may be processed at the same time, so it will be necessary to compute one IFFT per OFDM symbol. The IFFT was defined previously on Eq. (2.2). It is important to remark one more time that N is the FFT size, which is the next power of two of the number of active subcarriers. As it is commonly known, doing the calculation with a power-of-two number of points is more efficient, and this technique has been adopted in every design. It is important to mention that for doing these calculations the library *numpy* has been used, which provides a really good performance in digital signal processing commands. What is more, the reader should remember that the number of symbols is influenced by two terms: the number of active subcarriers, e.g, the total resources in terms of symbols, but also by the number of slots, where 14 OFDM symbols represent 1 slot. Therefore, now you can think that a good way to program this action is a for loop, where an IFFT is performed for every OFDM symbol. But that is not the best way of doing that. *Numpy* allows to execute more than one IFFT operation in parallel, we have to store the data in a matrix, where the number of columns specifies the number of IFFT operations. This is helpful and improves the quality of the code since we avoid using for loops that makes always this kind of software slower.

The output of this stage is the signal in the time domain, with the frequency sample required except for the last part of the oversampling, if it applies. But the signal is not ready to be transmitted over the air yet. There are still some steps that are explained next.

Cyclic prefix

Once the signal has been transformed into the time domain, the cyclic prefix must be added. As it was already specified in (2.5), the last N_{CP} samples of every symbol must be extracted and added to the beginning of the symbol. This has to be done for every symbol, taking into account that the number of samples extracted depends on the symbol position as well as the specified numerology. Now we do have to operate recursively because not every symbol has the same CP length. It was already commented in the last chapter, but in 5G NR every 7 symbols, there is one with a shorter duration. Insert the CP requires only one action: take the last samples of the signal in the time domain and put them at the beginning.

It is important to mention here that as we have commented, the oversampling changes the number of subcarriers and therefore the number of points of the IFFT. It means that now the number of samples that the cyclic prefix must add is different than the number if no oversampling is applied. The number of samples of the cyclic prefix was originally defined in (2.5). Now we must add the same factor that changes the number of subcarriers, e.g, the value of oversampling in frequency domain. If we do not do this, the percentage of samples that contains the cyclic prefix will not meet the standard requirements.

Peak clipping

Flexible numerology that 5G introduces exhibits a high peak-to-average power ratio (PAPR), which increases the demand on the performance of radio frequency (RF) power amplifiers [Gomes et al., 2018]. For this reason, a peak-clipping stage is introduced to reduce the PAPR of the signal before it is filtered [Rateb and Labana, 2019, Levanen et al., 2017]. The peak clipping function is defined as

$$G(x(n)) = \begin{cases} x(n), & |x(n)| \leq A \\ A \cdot e^{j \cdot \angle(x(n))}, & |x(n)| < A, \end{cases} \tag{3.4}$$

where $x(n)$ is the input signal, and A a scale factor based on a PAPR target in decibels, $\text{PAPR}_{\text{target}}$. The scale factor is obtained as

$$A = 10^{(\text{PAPR}_{\text{target}} - \text{PAPR}_x)/20}, \tag{3.5}$$

where PAPR_x is the PAPR of the signal input x in decibels, that is defined as follow,

$$\text{PAPR}_x = 20 \log_{10} \left(\frac{\max(|x(n)|)}{\text{rms}(x)} \right), \tag{3.6}$$

being rms(x) the root mean square value of $x(n)$.

The purpose of peak clipping is to reduce the amplitude of some samples in the time-domain signal. The function does not disrupt the sample if the absolute value of the sample is lower than the parameter A. If it is greater, it does not disrupt the phase, changing the module by the value of the parameter A. Doing that, the phase remains the same and therefore the signal distortion is minimized.

Spectrum shaping filter

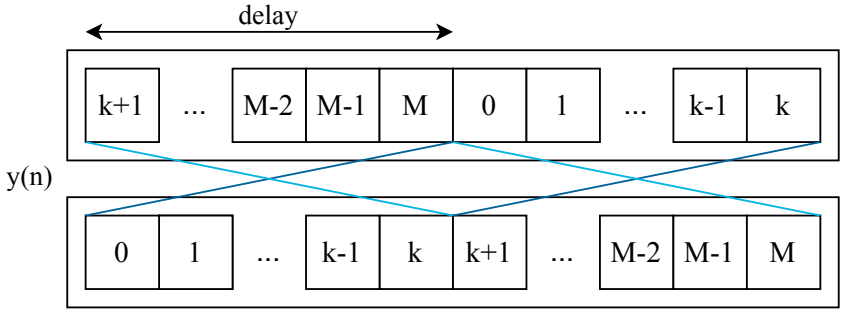

Figure 3.5 Due to the filter delay, it is necessary to reorder the output signal.

The addition of the CP will distort the spectrum and it might now contain some spurious radiation beyond the bandwidth channel. Therefore, it is necessary to filter the signal with a low-pass filter before modulating the carrier and radiating the resulting RF signal over the air [Park et al., 2019]. This low-pass filter is aimed at shaping the RF signal spectrum, that is why this stage is referred to as spectrum shaping [Cheng et al., 2016].

The low-pass filter has been designed using the Kaiser window $h(n)$ [Oppenheim et al., 2001], that is defined as

$$h(n) = \begin{cases} \frac{I_0\left[\beta(1-[(n-\alpha)\alpha]^2)^{1/2}\right]}{I_0(\beta)}, & 0 \leq n \leq N, \\ 0, & \text{otherwise}, \end{cases} \tag{3.7}$$

where $\alpha = N/2$, being N the length of the filter, i.e, number of taps, and $I_0(\cdot)$ represents the zeroth-order modified Bessel function of the first kind. Lastly, β and N are design parameters that we have to obtain in order to design our filter according to the specifications. As a design parameter, we want at least $A = 100$ dB rejection in the stopband, being the cutoff frequency (F_c) of the filter the bandwidth of the generated OFDM signal divided by two (we are working with the equivalent low-pass signal), given by

$$F_c = \frac{\text{BW}_{\text{signal}}}{2} = \frac{N_{\text{RB}} \cdot \Delta f}{2}. \tag{3.8}$$

The transition bandwidth is the difference between the channel bandwidth and the cutoff frequency, i.e., $\text{BW}_{\text{tr}} = \text{BW}_{\text{ch}} - F_c$. The value of BW_{ch} for the proposed parameters is obtained from Table 2.2. Next step is to get the parameters needed for the filtering, β and N, that are obtained with the following equations,

$$\beta = \begin{cases} 0.1102(A-8.7), & A > 50, \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 \leq A \leq 50, \\ 0, & A \leq 21, \end{cases} \tag{3.9}$$

and

$$N = \frac{10^{A/10} - 7.95}{2.285(\pi \cdot \text{BW}_{\text{tr}})} + 1. \tag{3.10}$$

Once the desired filter has been attained, the next step is to calculate the filter output. We have used the FFT to compute circular filtering. Hence,

$$Y(f) = X(f) \cdot H(f). \tag{3.11}$$

However, it is necessary to do one more step to get our filtered signal. The filter delay is $k = N/2$ samples, which means that the first samples of the output are allocated from sample $N/2$ to the end. Moreover, since we have applied circular filtering, the last samples will be allocated at the beginning of the obtained signal. To solve that, it is necessary to reorder the samples, as shown in Fig. 3.5.

In practice, the choice of F_c must be done carefully because a low value of the cut-off frequency might involve in a distortion of the received constellation. In other words, it is a trade-off between distortion in the constellation and spectrum spurious due to the CP. It is possible illustrate this with an example. A low value of F_c means that part of the signal will be outside the filter borders, as shown in the left pictures on Fig. 3.6. In that case, the F_c was equal to the 97.5 % of the signal bandwidth. For the second case, the value of F_c was set to the 102.5 % of the bandwidth signal. As it can be shown, that adjustment was doing manually, running the simulation and observing which value was okay. Finally, the value that was set for the official version was the second one, where the constellation received looks much better than in the first case, where it is shown that the signal spectrum is completely distorted.

3.2 Code availability

One of the purposes of this project is to make the software publicly available. This work is not only intended for private use, the tool must be available to download for everyone. If we are in an engineering context, it is

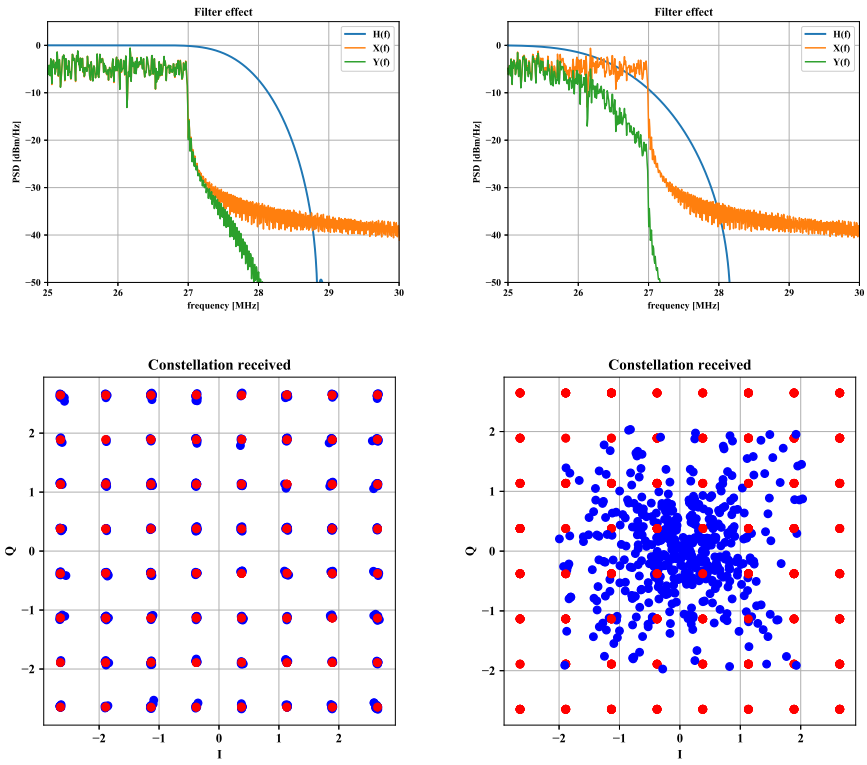

Figure 3.6 Spectrum shaping filter: how important is the choice of cut-off frequency. 97.5 vs 102.5.

obvious that GitHub is the most used online repository. But as you know, this tool is developed in Python, and we must not forget that there is an official repository for this programming language. Lastly, this work is planned to be submitted as a paper in the IEEE Access [1]. There is also a tool for publishing code in a research context called Codeocean. Without any doubt, this tool is also an important target that must be taken into account.

Before start commenting on how the code is available in every platform, it is important to remark how the code is structured into files. It consists of only 3 files:

- **NRsim.py** (See Code A.1). It includes all configuration variables commented during this chapter and the simulations that can be performed that are discussed in the next chapter. This script can be considered as an example of use, and it is not part of the tool itself.
- **NRlab.py** (See Code A.2). This file contains all the functions related to the multicarrier generation, OFDM transmitter and receiver, and all elements of the block diagram that is represented on Fig. 3.3. It is considered the main part of the project.
- **NRutils.py** (See Code A.3). Here is where all extra code that is necessary to run the main applications is located. It contains also some functions related to modulation/demodulation, filtering, etc.

With all these ingredients, what was clear was that this software must be published. Finally, all of the options were chosen and the code is publicly available in all of the platforms that we have commented. The process to put the code available in every context and also how to set up the project are discussed next.

3.2.1 Codeocean

Codeocean is a platform where users can publish code for free. As they say on his website, their mission is to make computational research easier, more collaborative, and durable. A big step was done when in 2017 they partnered the IEEE. It enabled the author to enhance the visibility of their papers, going one step further and

[1] *https://ieeeaccess.ieee.org/*

allowing everyone not only to discover the project reading the paper, but also trying a demo running code and exploring how it is done in terms of code.

Since the first moment that this project was planned to be published, the intentions were to publish the code also in this platform. Finally, we succeed in the publication and this project is publicly available since June 2020 at Codeocean [Palomino et al., 2020]. The fact of upload a project to Codeocean means that a Digital Object Identifier (DOI) number is assigned, meaning the project easier to be referenced. The DOI number that was assigned to this project was: *10.24433/CO.2712496.v1*.

Figure 3.7 Screenshot of my capsule at Codeocean.

Every project in Codeocean is associated with a capsule, that is a platform where the user can observe the project from outside it: it is possible to run the code but can not modify anything. It is also possible to open output files but the access is limited in some cases by the administrator. Our capsule in Codeocean is available in the following link: *https://codeocean.com/capsule/7434591/tree/v1*. If you click on it, you will see something similar like the screenshot represented in Fig. 3.7. On the left side of the screen, you will see the working directory as well as some files under the folder called *code*. That folder is where the code is placed and you can open the files and take a look at the Python scripts. However, if you try to edit the files you will see that it is not possible, since it is blocked by the user. That is the concept of a capsule that was commented above. On the right-top part of the web page, there is a blue button where the user will run the app. This will generate some messages on the log console but also will create the output files that will appear under the output folder. In this particular case, the user will find all waveforms generated and the results of every simulation.

As was said, this project was approved on the 14th of June by the Codeocean reviewers and it is available for everyone. As of the day, I am writing these lines, July 2020, this is the only project uploaded to Codeocean related to 5G for educational purposes.

3.2.2 PyPI

The tool that is presented in this work would be ideally preferable to be used by other scripts or functions. In the end, it is intended for the understanding of 5G-NR waveforms but it can be also helpful if you need to test your application and need to generate 5G-NR signals. Therefore, it is important to look for something that allows you to import the code into your application without any problems. The result of that is the necessity of upload this project to a Python repository: the Python Package Index (PyPI).

It is estimated that more than 230 thousand packages are available to download in PyPI. It is considered the official repository of this programming language. For that reason, it was considered a good idea to upload this project to PyPI. The difference here is that now the reason why people would download this package is a

different one than the reason they had with Codeocean. Now we are not talking about researching, but about professional purposes. Therefore, file NRsim.py is not included here.

The project was successfully uploaded to PyPI in June 2020 and can be found on the following link: *https://pypi.org/project/NRlabdtsc/*. It is also important to mention that it can be installed easily in our Python environment just by executing the following command:

```
>> pip3 install NRlabdtsc
```

Once the installation is complete, you can import the package with the following lines at the beginning of your script:

```
>> import NRlabdtsc.NRlab as NR
>> import NRlabdtsc.NRutils as utils
```

After that, you can code whatever you want using this package. It is important to mention that once we have downloaded the package, we can import it on the simulation script, NRsim.py, instead of having every file in the current folder. Now the package is imported in the whole system, and it is possible to check that the package is installed by executing the following command:

```
>> pip freeze
```

This command will return every package installed in the system sorted alphabetically. The package NRlabdtsc will be on the list, with the last version available when the user downloaded it.

3.2.3 Github

Lastly, this project is also uploaded to Github, the most famous repository for collaborative documents. Github also supports Git, a free distributed version control system, so it is possible to track different versions of the code, which is helpful to have everything well organized. The installation of the repository can be done by executing the following command:

```
>> git clone https://github.com/guipalloz/NRlab-dtsc.git
```

There is nothing much that can be said here since the author has worked before with this tool and has experience on it, therefore no extra time has been spent on learning this tool. The repository is available at the following webpage: *https://github.com/guipalloz/NRlab-dtsc*.

4 Results and discussions

Scientists investigate that which already is; engineers create that which has never been.

Albert Einstein

The last chapter of this thesis covers all the simulation carried out, and their following results and discussions. As it has been mentioned during this document, the purpose of all this work is to provide a complete tool that allows the users to create their experiments. However, as an example, as well as part of the validation process, some simulations have been developed that will be explained during this chapter. As a consequence, with those simulations, some interesting results can be extracted. Most of the results have indeed been already commented on during the whole document, but here we are describing, step by step, which simulations are important to validate this work and how the user can get some inspiration on these simulations and perform new ones.

Since this is the last chapter of this thesis, there is a section that comments a few lines about how was the methodology that was followed and, most important, what are the conclusions of the works as well as some future lines. This section can be found at the end of this chapter.

4.1 Simulations and results

To measure the system performance of the proposed tool, this section is organized as follows. Firstly, the result of the multicarrier generation stage, explained in detail during chapter number 3, is analyzed. More precisely, the 5G signal is generated in the time domain. Then, a simulation through an AWGN channel is performed, and its result is presented and discussed. Here the main figures of merit of a classical digital communication scheme will play an important role. We will calculate some parameters and it will be shown how good our software works. Finally, the generated signal is sent to an external server that creates the signal and tests it with a power amplifier located in the University of Chalmers, Gothenburg (Sweden). All simulations explained during this chapter have been implemented and the code is included at the end of this work (Code A.1).

4.1.1 Simulation number 1: Multicarrier generation stage

The first simulation that was performed was the direct output of the block diagram that was described in the last section. As was commented, the output was a signal generated in the time domain. Hence, an analysis is performed to study this output in detail. The problem of the PAPR in 5G is also addressed here, where a study of how PAPR affects to our system and how good it is improved when peak-clipping techniques are introduced (which also motivates and validates the assumption it was done by introducing this element) has been performed and their results are included also next.

Analysing the signal in the time domain

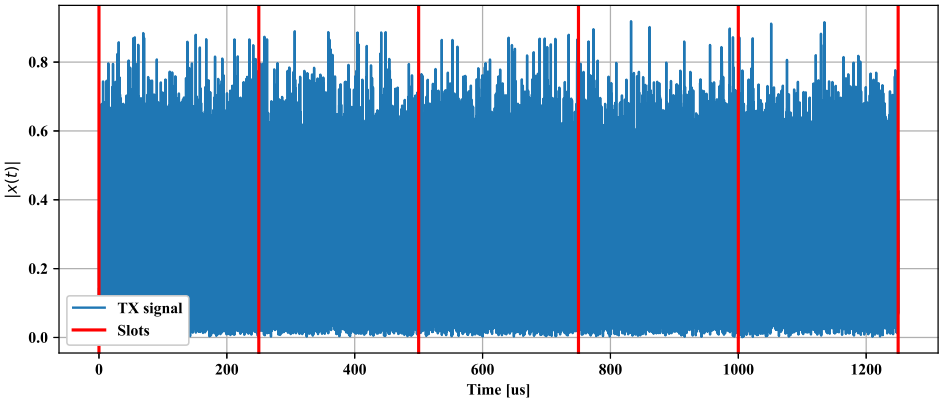

Figure 4.1 5G waveform for a subcarrier spacing equal to 60 kHz and 5 slots.

The output of the multicarrier generation stage is the generated 5G signal in the time domain, which was shown in Fig. 4.1 for 20 RBs, a subcarrier spacing of 60 kHz and 5 slots. As was already mentioned, those parameters are just an example, and the software allows us to change them. It is possible to appreciate how the time duration of the signal corresponds with the parameters given in Fig. 2.1, where all numerology was represented. To give an example, a subcarrier spacing equals 60 KHz refers to the third row on the figure. It is possible to see on the picture that 56 symbols are equal to 1 subframe (1 ms), therefore the duration of each symbol is $1/56$ *ms*. 14 symbols are 1 slot, which means that in the simulation we have $14\cdot = 70$ OFDM symbols. Lastly, the duration of every symbol is $1/56\ ms \cdot 70 = 1.25\ ms$ that is exactly the duration of the signal represented in Fig. 2.1.

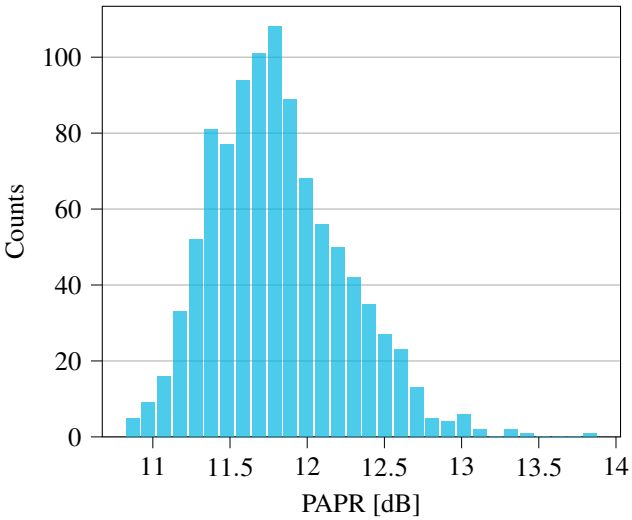

Figure 4.2 Histogram of the PAPR, obtained for 1000 signals generated.

Table 4.1 Numerology for PAPR simulation.

Number of RB	Δf	PAPR target	Number of simulations	Seeds
250	15 KHz	10.5 dB	1000	0-999

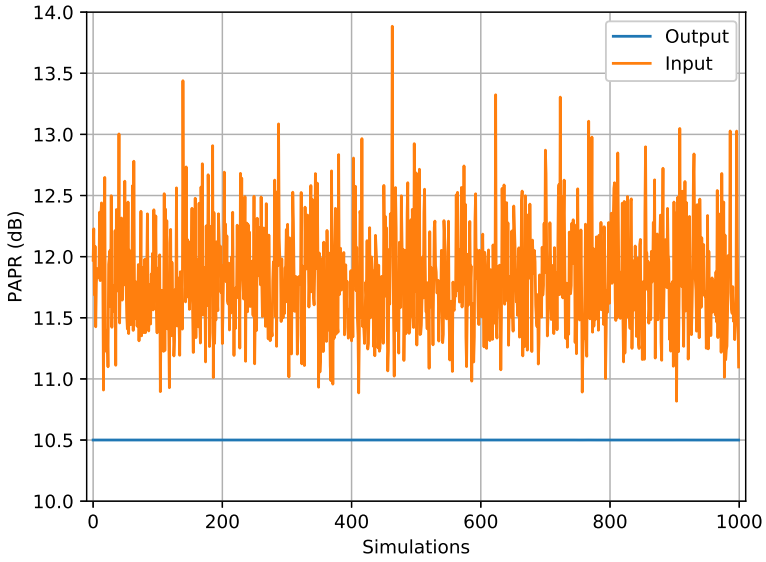

Figure 4.3 Input and output PAPR of the peak clipping stage along 1000 simulations.

Peak clipping stage motivation

Before finishing the transmitter stage, it was mentioned in the last chapter that a peak clipping was necessary to reduce the PAPR. The next experiment is to calculate precisely that PAPR and motivate somehow the fact of adding this element in the transmitter stage. The simulation generated a signal with the numerology specified in table 4.1. It consisted of generating 1000 times a signal with the characteristics mentioned and averaging the results of the PAPR obtained before and after the peak-clipping stage.

In Fig. 4.2 the histogram of the PAPR before peak clipping is shown. A histogram is defined as a representation of the distribution of some data. In this case, the horizontal axis represents the values of the PAPR found on the signal samples, and the vertical axis indicates the number of occurrences of those values in every sample. Analyzing the figure, most of the obtained PAPR for each simulation is higher than 11 dB, so it is evident that peak clipping is necessary [Ochiai and Imai, 2001]. Using the peak-clipping technique defined in Section 3.1.2 with a PAPR target of 10.5 dB, the obtained normalized mean squared error (NMSE) between the PAPR after peak clipping and the target value of 10.5 dB is equal to 0.023%. Besides, an average of 15.1 samples for every simulation was affected by clipping. One last aspect that can be analyzed is the one that is represented in Fig. 4.3, that is the PAPR calculated before and after peak clipping along 1000 simulations. It is shown how the improvement is performed, with an almost perfect output PAPR equals to the target one, that was 10.5 dB. The script that performs all these techniques mentioned here can be found in the Code A.4.

In this tool, a basic peak-clipping function has been implemented (see Section 3.1.2), but more advanced techniques can be applied here, which reduces, even more, the PAPR and minimizing residual effects. In parallel to this project, another thesis has done a deep study about peak clipping techniques [López, 2020], therefore it could be interesting to have performed an analysis of how different techniques affects the waveform generation. Without any doubt, that analysis will be useful to upgrade this software in the future.

4.1.2 Simulation number 2: AWGN channel and most typical figures of merit

The challenge of this second simulation that is presented in this chapter is to obtain the classic figures of merit that are calculated when a digital wireless system is tested. The starting point will be the output of the transmitter stage, as we did with the last simulation. Two measurements have been specified: Bit Error Rate (BER) and Error Vector Magnitude (EVM). Both experiments were performed following the parameters specified by the 5G-NR standard.

Bit Error Rate (BER)

To obtain the performance of the designed tool in terms of BER, the waveform generated was passed through an additive white Gaussian noise (AWGN) channel, and then it was detected. It is important to perform a BER simulation analysis in every digital system because those parameters show us how is the quality of the full radio link in terms of bits. BER value is extracted for specific transmission conditions, i.e., for a given value of E_b/N_0 and a modulation order. It is possible to perform a Montecarlo simulation that iterates over a range of values of those. More precisely, in this simulation, the values selected for E_b/N_0 were from 0 dB to 15 dB. Additionally, several modulation techniques were implemented to evaluate the performance of the proposed model. Precisely, modulations supported by 5G such as QPSK, 16QAM, 64QAM, and 256QAM were chosen. The random bits were modulated and inserted in each subcarrier following the order modulation specified. The result of the Montecarlo simulation is shown in Fig. 4.4. To calculate the BER after the transmission and its posterior reception, it is just dividing the bits that were different after the detection by the total number of bits transmitted. If we do that for every value of E_b/N_0 a curve will be obtained. If we repeat the same procedure but now for every other order modulation, a similar figure like the one represented here will be obtained. But to know if the results are correct or need to be improved, it was also considered to plot the theoretical curves. However, there does not exist a close expression of those curves. As a consequence, the dashed lines represent the maximum level that the BER can be. In mathematical terms, those curves can be approximated as

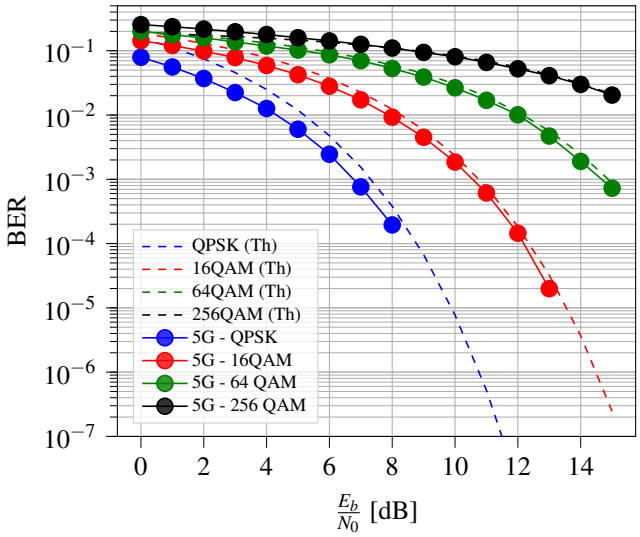

Figure 4.4 Average Bit Error Rate (BER) versus a sweep in the bit energy to noise power ratio.

$$BER \approx \frac{4}{\log_2 M} Q\left(\sqrt{\frac{3E_b/N_o \log_2 M}{M-1}}\right). \tag{4.1}$$

From Fig. 4.4 can be observed that the simulated curves are below the theoretical ones since these latter were an approximation of the first ones, as was said in last above. Also, for a fixed value of E_b/N_0, the simulation with a lower order modulation results in a lower BER. However, the number of bits that are inserted into the subcarriers is also lower, so the bit rate is reduced. This means that the choice of the order modulation relies on the channel conditions: a higher data rate will be preferable, but the BER value must be suitable for the specific desired application.

Error Vector Magnitude (EVM)

Another typical figure of merit of a digital system is the error vector magnitude (EVM), which measures how far are the constellation points from the ideal one. The EVM is calculated as follows,

$$\%\ \text{EVM} = \frac{\sqrt{\frac{1}{N}\sum_{n=0}^{N-1} \text{I}_{\text{err}}^2(n) + \text{Q}_{\text{err}}^2(n)}}{\text{EVM Reference}} \times 100\%, \tag{4.2}$$

where n is the symbol index, N the number of symbols for the given transmission, and I and Q are the in-phase and quadrature components of the error vector, where Ierr is defined as the difference between de original points in the constellation and the received ones. Lastly, EVM Reference is a normalization factor that is calculated as

$$\text{EVM Reference} = \max \sqrt{\left(\text{I}_{\text{err}}^2(n) + \text{Q}_{\text{err}}^2(n)\right)}, \tag{4.3}$$

i.e., the maximum value of the norm of every point of the constellation. The Montecarlo simulation that was carried out was exactly the same it was done in last simulation but now calculating the EVM instead. Therefore, we can extract a similar figure and it is expected that the system performance would improve with higher values signal to noise ratio. The results of the simulations are shown in Fig. 4.5 and it is possible to appreciate that the performance tendency is quite the same we had for the BER simulation.

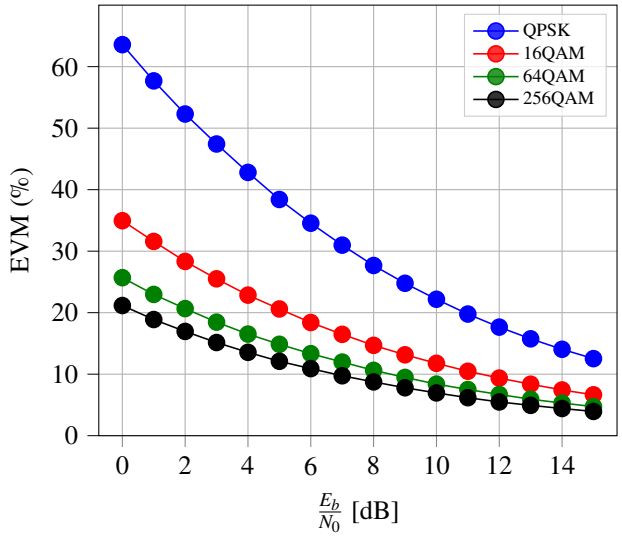

Figure 4.5 Error Vector Magnitude (EVM) versus a sweep in the bit energy to noise power ratio.

4.1.3 Simulation number 3: Sending the generated signal to an external RF WebLab

The last simulation consists in test our system in an external environment that is commonly used by the research group that manages this project. This simulation is organized into two sections: firstly, the external laboratory is introduced, and the simulation parameters are explained. Lastly, the results of the simulations are presented.

RF Weblab at Chalmers University

One last simulation was performed, to give compatibility with the work performed by the Digital Pre-Distorsion (DPD) research group that will use the tool. This simulation consists of sending the signal to an RF WebLab located at Chalmers University of Technology [Landin et al., 2015]. It is possible to remotely access the measurement system that was initially designed for hosting the IMS Digital PreDistortion Student Design Competition in 2014 and 2015. According to the RF WebLab webpage, the intentions are "to give DPD developers worldwide access to a common amplifier and measurement system, making the evaluation

of the performance of DPD algorithms easy, and enabling comparisons between various algorithms on a common platform". The measurement setup consists of the following:

- A Vector Signal Transceiver (PXIe-5646R VST) with an instantaneous bandwidth equals to 200 MHz.
- A linear driver amplifier with a gain approximately equals to 40 dB.
- The DUT which is a nonlinear GaN power amplifier (Cree CGH40006-TB).
- The DUT has also a power source module (PXI-4130) that allows the users to know the power amplifier current consumption.

However, there are also some limitations that the signal must meet before taking the measurements. To give some examples, the signal that is interpreted as the input must have a PAPR lower than 20 dB, a maximum number of samples of 1 million, and a maximum peak power of -8 dBm. It is also important to remark that the peak output power level is limited to 6 W. Also, the sampling rate must be 200 MHz. If you have read the last chapter, where all elements of the transmitter stage were explained, you will understand now how important is to add the block that performs an oversampling that changes the sampling rate. Thanks to that, we can easily generate a custom signal for every application.

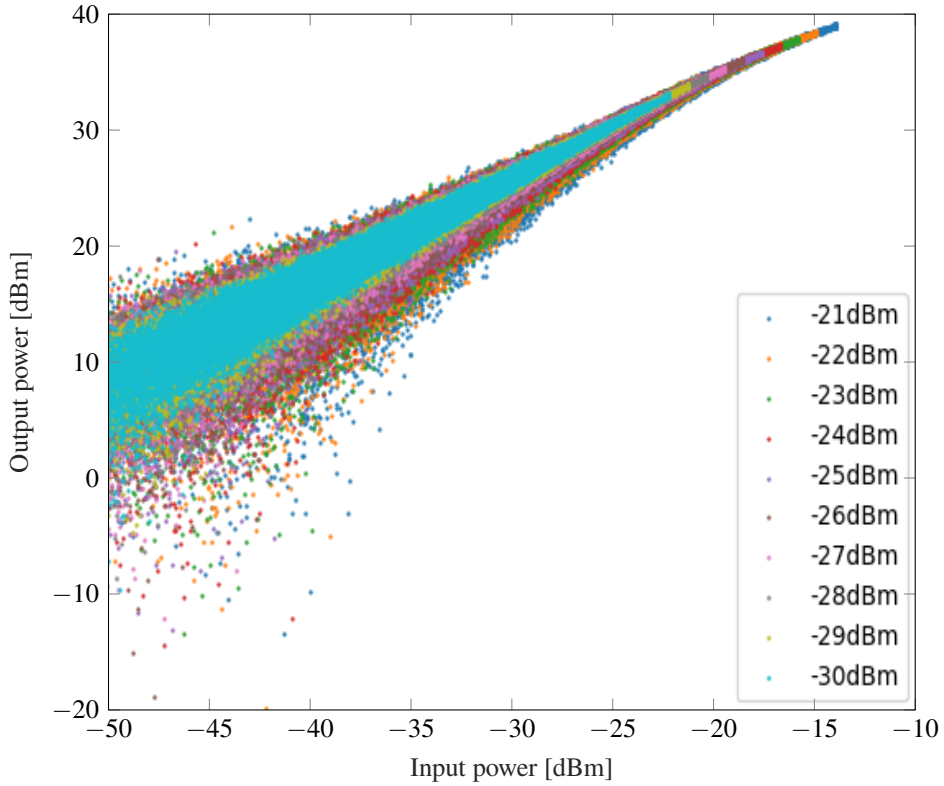

Figure 4.6 AM-AM characterization of the power amplifier.

Programming the script and simulation

RF Weblab provides some scripts in Matlab [1] that allows users to access the power amplifier. However, the purpose of this simulation is to get the measurements without using any external actions. Hence, it was convenient to explore other techniques that integrate those files in our simulations. Fortunately, Matlab has an engine that allows us to run Matlab scripts with Python. To install the engine, all we need to do is to follow these steps:

1. Open Matlab and execute the following command:

[1] scripts available at: *http://dpdcompetition.com/rfweblab/access-details/*

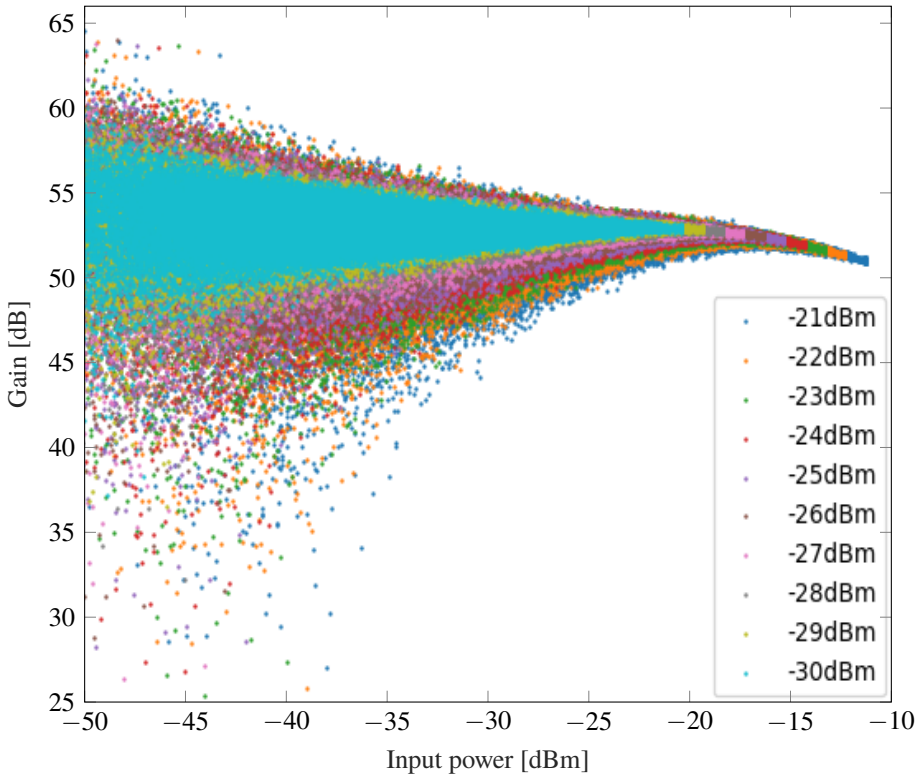

Figure 4.7 Gain-AM characterization of the power amplifier.

```
>> matlabroot
```

The output will be the path where Matlab is installed. Navigate to that folder.

2. Now open a Windows prompt execute the following command:

   ```
   >> cd matlabroot/extern/engines/python
   ```

   where *matlabroot* is the output of the first step.

3. Lastly, you will find a Python script. Execute it running the following command:

   ```
   >> python3 setup.py install
   ```

   It will install the MatLab engine for Python.

4. After that, the Matlab engine for Python will be ready, and we can load it in Python with the following line: `eng = matlab.engine.start_matlab()`. It will return an object, and we can call every Matlab command or function as a method of this object. It is also possible to use arguments, just add them as parameters when the method is called.

The experiment setup is included in the simulation script already commented, located at the end of this work (Code A.1). It is possible to appreciate how a flag called *RunMatlabFunction* enables this third simulation. If the value is true, then some data are saved into a mat file that the MatLab script will use. The Matlab script we are using is similar to the ones available at RF Weblab, but we are loading the data in Matlab instead of creating them. More precisely, we are generating 10 different signals sweeping RMS values, from -30 dBm to -21 dBm. The obtained results are then analyzed and the gain in decibels for each sample is obtained. For every RMS value, several simulations are performed, and then the result is averaged. The results are therefore with less noise but it will take more time to get the results. As always, it is a trade-off.

Once the Matlab script has finished and the measurement is obtained, it is possible to load again the output data in Python. Furthermore, we obtain the input and the output of the RF Weblab (both real and imaginary parts). And interesting and direct results is to calculate the AM-AM and Gain-AM characterization, very useful when studying non-linear behaviors. The AM-AM characterization represents the amplitude variation

of the output respect to the input, whereas the Gain-AM represents the power amplifier gain (the output minus the input, if measured in decibels) respect to the same input. Both characterizations have been calculated and the resulting curves are represented in Figs. 4.6 and 4.7. The script that generates the figures is attached and can be found in Code A.5.

4.2 Conclusions and future lines

The complete understanding of OFDM system, the processes involved in 5G waveform generation, and the performance methods that are state of the art are concepts that need to be taught in Electrical Engineer programs. To help Electrical Engineering students from all over the world to help to understand those concepts, this work has been completely developed for waveform generation categorized as a FOSS platform, which means that it is an opensource platform. This software allows the student to observe and study how every block of the transmitter and reception stage works in 5G in terms of digital communication, looking at them not only as a black-box but also being able to analyze how they are implemented. Additionally, the platform gives the students the opportunity of modifying design parameters and see how they affect the transmission in terms of PAPR, BER, and EVM, which are the most typical figures of merit. The students observe a practical example that links with the theoretical contents already studied in the program, hence it helps them to acquire the desired knowledge.

Thinking about myself, this work has been useful to reinforce my knowledge in mobile communications. I had previous knowledge in 5G NR, but developing this tool in a real environment and generating waveforms according to the standard gave me practical experience in this field. What is more, it is always good to develop an open-source platform, because everyone can read what you have done and if they want, they can modify and improve your work. That is how engineering works and how humanity has reached impossible goals: helping each other. Additionally, this master thesis somehow concentrates all work last year. This document reflects how was the process of this project, starting from the theory, and finishing coding and testing.

But this project does not end here. This thesis is just the first step that summarized the first version of this software. After the project dissertation, this tool will be sent to the department responsible for this project and they will be able to continue improving this tool. Furthermore, this work belongs to a tendency observed during the last years: the migration of all research projects to Python. As a consequence, this is not the only thesis that presents a work programmed in Python, and in the future, there will be an integration of all of them.

As it has been commented previously, the intentions with this project were to submit a paper for publication on IEEE Access magazine, so the works do not end here. If the request is accepted, a reviewing process will start, therefore it is probable that the project will be modified in the future.

To sum up, the author feels satisfied with this project. He has developed a complete tool from nothing to what is published on Codeocean, and he has given to everyone the possibility to study and explore the different waveforms that can be generated according to the standard that will define the way that cellphones will share data in the next generation of mobile communication.

Appendix A
Implementation of the project

A.1 Simulation script that uses the toolbox presented in this work

Code A.1 Simulation script that uses the toolbox presented in this work (`NRsim.py`).

```
his#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Author: Guillermo Palomino Lozano
Organization: University of Seville
Date: May 2020
"""
import numpy as np
import os
import matplotlib.pyplot as plt
import scipy.io as sio
#import matlab.engine
from matplotlib.ticker import ScalarFormatter
import warnings

import NRlabdtsc.NRlab as NR
import NRlabdtsc.NRutils as utils

#------------------------------------------------------------
plt.close('all')

NRB = 150 # Number of resource blocks
Df = 30e3 # Subcarrier Spacing
OvS = True # Oversampling
Fs = 200e6 # Only important when oversampling is true
Nslots = 1
FR = 1 # Frecuency range
M = 64 #Modulation order
PAPR = 10.5 #PAPR desired
SEED = 1000 # Seed for random number generation
PlotOptions = True #True for active plots (Subcarriers Tx/Rx, Constellation,
    Spectrum, etc )
#-------------------------------------------------Pl-------------
#   Simulations available:
```

```
BERsimulation = False
Amplifiersimulation = False
RunMatlabFunction = False
Filtering = True

#------------------------------------------------------------
warnings.filterwarnings('ignore')
if (BERsimulation == True):
    #Modulation = np.array([4])
    Modulation = np.array([4, 16, 64, 256]) # Modulation order
    EbNo = np.arange(0,15,1)
    BER = np.zeros((len(EbNo),len(Modulation)))
    errorteorico = np.zeros((len(EbNo),len(Modulation)))
    EVM = np.zeros((len(EbNo),len(Modulation)))
    NMSE = np.zeros((len(EbNo),len(Modulation)))
else:
    Modulation = np.array([M]) # Modulation order
    EbNo = np.array([float('inf')])
    BER = 0
for jj in range(len(Modulation)):
    M = Modulation[jj]
    k = np.log2(M)
    MyStructure = NR.Numerology(Df,FR)
    print("\n------------- STARTING TX -------------")
    print("\n M = \%d" \%(M))
    x, Bn, FFTSize, Alphabet, alpha, Eb, const_points, Fs, Fo, BwCh, PAPRx,
        PAPRy = NR.MulticarrierGeneration(MyStructure, NRB, OvS, Fs, Nslots, M,
         PAPR, SEED, PlotOptions, Filtering)
    #Ready for real measurements!
    if (RunMatlabFunction == True):
        #RunBERsimulation.m implementation goes here
        Nmeas = 3
        Pin = np.arange(-30,-20)
        #Pin = np.array([-30,-20])
        X_amp = np.zeros((Pin.size,x.size),dtype=np.complex128)
        Y_amp = np.zeros((Pin.size,x.size),dtype=np.complex128)
        currpath = os.getcwd()
        os.chdir("Chalmers")
        print("\n------------- STARTING Power Amplifier stage -------------")
        print("Starting Matlab engine...")
        eng = matlab.engine.start_matlab()
        for ii in range(len(Pin)):
            RMSin = Pin[ii]
            print("\n Saving signal for real measurements...")
            sio.savemat('NR_signal.mat', {'x':x,'RMSin':RMSin,'Nmeas':Nmeas})
            print(" Running real measurement in the RF WebLab for Pin = \%d dBm
                ... it will take around \%d minute(s) ..." \% (int(RMSin), int(
                Nmeas)))
            #os.system("NRChalmers.exe")
            eng.NRChalmers(nargout=0)
            print(" done!")
            mat = sio.loadmat('NR_signal_Output.mat')
            aux = mat["x"][:,0]
            X_amp[ii,:].real = mat["x"][:,0].real
            X_amp[ii,:].imag = mat["x"][:,0].imag
            Y_amp[ii,:].real = mat["y"][:,0].real
            Y_amp[ii,:].imag = mat["y"][:,0].imag
```

```
        #data = mat["data"]
    #Save the results in a file
    eng.quit()
    os.chdir("../")
    np.savez('PowAmp.npz',X_amp=X_amp,Y_amp=Y_amp,Pin=Pin)
    y_amp = Y_amp[1,:]
    print("\n------------- FINISHING Power Amplifier stage -------------")
if (BERsimulation == False):
    print("\nGenerated signal:")
    print(x)
    print("\nBits transmitted:")
    print(Bn)
print("\n------------- STARTING RX -------------")
for ii in range(len(EbNo)):
    if (Amplifiersimulation == True):
        data = np.load('PowAmp.npz')
        Xamp = data['X_amp']
        Yamp = data['Y_amp']
        Pin = data['Pin']
        Nsim, Nsamples = Yamp.shape
        for jj in range(Nsim):
            print("\n Analyzing measurement for Pin = \%.2f dBm" \% Pin[jj])
            y = NR.Channel(Yamp[jj,:], EbNo[ii], Chtype="AWGN")
            rx_Bn, rx_const_points, ACPR, ACPR2 = NR.MulticarrierReceiver(
                NRB, MyStructure, y, int(Nslots), Alphabet, alpha, int(
                FFTSize), Fs, Fo, PlotOptions, BERsimulation,BwCh, Filtering)

            utils.DrawConstellation(const_points.flatten(), rx_const_points.
                flatten(), NUMPOINTS = 500)
    else:
        SNRdB=EbNo[ii]+10*np.log10(k)+10*np.log10(NRB*12/FFTSize)
        y = NR.Channel(x, SNRdB, Chtype="AWGN")
        if (BERsimulation == False):
            print("Received signal:")
            print(x)
        rx_Bn, rx_const_points, ACPR, ACPR2 = NR.MulticarrierReceiver(NRB,
            MyStructure, y, int(Nslots), Alphabet, alpha, int(FFTSize), Fs,
            Fo, PlotOptions, BERsimulation,BwCh, Filtering)
        if (PlotOptions == True):
            utils.DrawConstellation(const_points.flatten(), rx_const_points.
                flatten(),NUMPOINTS = 500)
        if (BERsimulation == False):
            print("Received sequence:")
            print(rx_Bn)

        if (BERsimulation == True):
            print(" EbNo = \%d dB" \% (EbNo[ii]))
            BER[ii,jj] = np.sum((Bn != rx_Bn))/Bn.size
            #print("\nBER para SNR = \%f dB: \%f" \% (EbNo[ii], BER[ii,jj]))
            ebno = 10**(EbNo[ii]/10)
            errorteorico[ii,jj] = 4/k*utils.Qfunct(np.sqrt(3*ebno*k/(M-1)))
            #arg = np.sqrt(3*k/(M-1)/No)
            #errorteorico[ii,jj] = (4/k)*(1-1/np.sqrt(M))*Qfunct(arg)
            EVM[ii,jj] = utils.EVM_Calculation(const_points.flatten(), rx_
                const_points.flatten())
            NMSE[ii,jj] = utils.NMSE_Calculation(x,y)
print("\n------------- FINISHING RX -------------")
```

```
if (BERsimulation == True):
    plt.figure()
    plt.semilogy(EbNo, errorteorico[:,0], '-',label='M = 4 (Th)', color='blue')
    plt.semilogy(EbNo, errorteorico[:,1], '-',label='M = 16 (Th)', color='red')
    plt.semilogy(EbNo, errorteorico[:,2], '-',label='M = 64 (Th)', color='green
        ')
    plt.semilogy(EbNo, errorteorico[:,3], '-',label='M = 256 (Th)', color='
        black')
    plt.semilogy(EbNo, BER[:,0],'-o', label='M = 4', color='blue')
    plt.semilogy(EbNo, BER[:,1],'-o', label='M = 16', color='red')
    plt.semilogy(EbNo, BER[:,2],'-o', label='M = 64', color='green')
    plt.semilogy(EbNo, BER[:,3],'-o', label='M = 256', color='black')
    plt.xlabel('EbNo [dB]'); plt.ylabel('BER');
    plt.grid(True,which="both",ls="-")
    plt.legend()
    axes = plt.gca()
    axes.set_ylim([10**(-7),0.5])
    plt.title('5G NR System performance for AWGN channel')
    plt.savefig('../results/berperformance.png')

    plt.figure()
    fig, ax = plt.subplots()
    for plt.axis in [ax.xaxis, ax.yaxis]:
        plt.axis.set_major_formatter(ScalarFormatter())
    plt.semilogy(EbNo, EVM[:,0],'-o',label='M = 4', color='blue')
    plt.semilogy(EbNo, EVM[:,1],'-o',label='M = 16', color='red')
    plt.semilogy(EbNo, EVM[:,2],'-o',label='M = 64', color='green')
    plt.semilogy(EbNo, EVM[:,3],'-o',label='M = 256', color='black')
    plt.xlabel('EbNo [dB]'); plt.ylabel('EVM (\%)');
    plt.grid(True,which="both",ls="-")
    plt.legend()
    plt.title('5G NR System performance for AWGN channel')
    plt.savefig('../results/evmperformance.png')

    plt.figure()
    fig, ax = plt.subplots()
    for plt.axis in [ax.xaxis, ax.yaxis]:
        plt.axis.set_major_formatter(ScalarFormatter())
    plt.semilogy(EbNo, NMSE[:,0],'-o',label='M = 4', color='blue')
    plt.semilogy(EbNo, NMSE[:,1],'-o',label='M = 16', color='red')
    plt.semilogy(EbNo, NMSE[:,2],'-o',label='M = 64', color='green')
    plt.semilogy(EbNo, NMSE[:,3],'-o',label='M = 256', color='black')
    plt.xlabel('EbNo [dB]'); plt.ylabel('NMSE');
    plt.grid(True,which="both",ls="-")
    plt.legend()
    plt.title('5G NR System performance for AWGN channel')
    plt.savefig('../results/nmseperformance.png')
else:
    EVM = utils.EVM_Calculation(const_points.flatten(), rx_const_points.flatten
        ())
    print("EVM: \%.2f" \% EVM)
```

A.2 Main 5G-NR toolbox script for waveform generation

Code A.2 Main 5GNR toolbox script for waveform generation (`NRlab.py`).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Author: Guillermo Palomino Lozano
Organization: University of Seville
Date: May 2020
"""

import sys
import os
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import NRutils as utils

plt.rcParams["font.family"] = "Times New Roman"

def Numerology(Df, FR):
    """
    Description
    ----------
    Specify the numerology according to the standard of 5GNR

    Parameters
    ----------
    Df : float
        subcarrier spacing.
    FR : int
        frequency range.

    Returns
    -------
    MyStructure : dictionary
        information about the numerology specified according to the standard.

    """
    TimeDomainStructure = [{"DF": 15e3,"Tslots": 1e-3, "FR": 1, "Range": [25,
        270], "mu": 0},
                           {"DF": 30e3,"Tslots": 0.5e-3, "FR": 1, "Range": [11,
                               273], "mu": 1},
                           {"DF": 60e3,"Tslots": 0.25e-3, "FR": 1, "Range": [11,
                               135], "mu": 2},
                           {"DF": 60e3,"Tslots": 0.25e-3, "FR": 2, "Range": [66,
                               264], "mu": 2},
                           {"DF": 120e3,"Tslots": 0.125e-3, "FR": 2, "Range": [32,
                                264], "mu": 3},
                           {"DF": 240e3,"Tslots": 0.0625e-3, "FR": 2, "Range":
                               [32, 264], "mu": 4}]
    Dfs = [15e3, 30e3, 60e3, 60e3, 120e3, 240e3]
    #Time-Domain structure definition:
    if Df == 60e3: #According to the standard: 2 possibilities with Df = 60 Khz
```

```
        if FR == 1:
            MyStructure = TimeDomainStructure[2]
        else:
            MyStructure = TimeDomainStructure[3]
    else:
        MyStructure = TimeDomainStructure[Dfs.index(Df)]
    return MyStructure

def MulticarrierGeneration(MyStructure, NRB, OvS, Fs, Nslots, M, PAPRd, SEED,
    PlotOptions, Filtering):

    Ncarr_act = NRB * 12 #Number of active carriers: Each RB means 12 carriers
    FFTSize = int(2**np.ceil(np.log2(Ncarr_act))) #Next power of 2
    if OvS == False:
        OvS = 1
        OvS_freq = 1
        Fs = float(FFTSize * MyStructure["DF"])
        Fo = Fs
    else:
        Fo = float(FFTSize * MyStructure["DF"])
        OvS = Fs / Fo
        if (OvS < 1):
            print("Error. OverSampling lower than 1")
            sys.exit(0)
        OvS_freq = np.floor(OvS)
        OvS_freq = 1
        # Fs is the value given at the input
        print(" Oversampling will be performanced: \%.2f" \% OvS)
    print(" From Fo = \%.2f MHz to Fs = \%.2f MHz " \% (float(FFTSize *
        MyStructure["DF"])/1e6, Fs/1e6))

    x, Bn, Alphabet, alpha, Eb, const_points, BwCh = tx_OFDM(NRB, Nslots,
        MyStructure, M, FFTSize, Ncarr_act, Fo, Fs, OvS_freq, SEED, PlotOptions
        )
    x = x - np.mean(x);
    x  = x  / np.max(np.abs(x));


    PAPRx = 20*np.log10(np.max(np.abs(x))/np.mean(np.abs(x))) # 10*log10(max(
        abs(x))/rms(x))
    print(" PAPR before clipping: \%f dB" \% (PAPRx))
    clip = 10**((PAPRd - PAPRx) / 20)
    idx = np.nonzero(np.abs(x) > clip)
    print(" \%d Samples will be affected by clipping" \% (np.size(idx)))
    x[idx] = clip*np.exp(1j*np.angle(x[idx]))
    PAPRy = 20*np.log10(np.max(np.abs(x))/np.mean(np.abs(x)))
    print(" PAPR after clipping: \%f dB" \% (PAPRy))
    if (Filtering == True):
            #do filtering stage here...
            x = utils.Filter_signal(Fs, BwCh*1e6, Ncarr_act*MyStructure["DF"], x
                )
    if (PlotOptions == True):
        plt.figure()
        n, bins, patches = plt.hist(x=np.abs(x), bins=20, color='#607c8e',alpha
            =0.7, rwidth=0.85)
        plt.grid(axis='y', alpha=0.75)
        plt.xlabel('Absolute value of the x(n) sample')
```

```
        plt.ylabel('Counts')
        plt.title('Histogram of absolute value samples of x(n)')
        plt.savefig('../results/histx.png')
        t = np.arange(x.size)* 1/Fs
        plt.figure(figsize=(10,4))
        plt.plot(t*10**6,np.abs(x), label='TX signal')
        plt.xlabel('Time [us]'); plt.ylabel('$|x(t)|$');
        plt.grid(True);

        for xc in range(0,Nslots+1):
            plt.axvline(x=(xc * MyStructure["Tslots"] * 1e6),linewidth=2, color
                ='r', label='Slots')
            if xc == 0:
                plt.legend(fontsize=10, loc='lower left')
        plt.savefig('../results/timex.png')
        f, Pxx = utils.PSD(x, Fs, FFTSize)
        Pxx  = 10*np.log10(Pxx) + 30
        plt.figure(figsize=(5, 4))
        plt.plot(f,Pxx)
        plt.xlabel('frequency [MHz]')
        plt.ylabel('PSD [dBm/Hz]')
        plt.show()
        plt.grid(True);
        plt.title('Power Spectral Density of the signal transmitted')
        plt.savefig('../results/spectrumtx.png')
    print(" Generated OFDM signal has \%d samples" \% (x.size))
    print(" FFTSize: \%d samples" \% (FFTSize))
    print("\n------------- FINISHING TX -------------")
    return x, Bn, FFTSize, Alphabet, alpha, Eb, const_points, Fs, Fo, BwCh,
        PAPRx, PAPRy

def tx_OFDM(NRB, Nslots, MyStructure, M, FFTSize, Ncarr_act, Fo, Fs, OvS_freq,
    SEED, PlotOptions):
    #Range of Channel Bandwidths for the Different Numerologies
    Df = MyStructure["DF"]

    if MyStructure["FR"] == 1:
        BwCh = np.array([5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100])
    else:
        BwCh = np.array([50, 100, 200, 400])
    #Checking that our NRB is in range
    if (NRB < MyStructure["Range"][0]) or (NRB > MyStructure["Range"][1]):
        print("The number of resource blocks does not match the defined
            numerology\n")
        print("Desired range: \%d-\%d \n" \% ( MyStructure["Range"][0],
            MyStructure["Range"][1]))
        print("Your value: \%d \n" \% (NRB))
        sys.exit(0)

    Ncarr_act = NRB * 12 #Number of active carriers: Each RB means 12 carriers
    print(" Number of active subcarriers: \%d" \% Ncarr_act)
    BwEff = Df * Ncarr_act
    BwEff = (BwEff + 1/2*(BwEff - NRB*12*Df-Df))/1e6 #Eq (18.1)
    #And now we allocate this BW into one of the possibilities defined above
    idxBw = np.nonzero(BwCh > BwEff)[0]
    Bw = BwCh[idxBw[0]] #Now we have set our final Bandwidth
    print(" Channel Bandwidth: \%d MHz" \% int(Bw))
```

```
print(" Bandwidth occupied: \%d MHz" \% int(Ncarr_act*Df/1e6))
NsymbOFDM = Nslots * 14 #14 OFDM symbols every time slot (see Figure 7.2)
Nres = NsymbOFDM * Ncarr_act #Total resources in terms of symbols
k = int(np.log2(M)) #Bits per symbol
Nbits = int(Nres*k) #Total number of bits
print(" \%d bits were generated" \% (Nbits))
np.random.seed(SEED)
Bn = np.random.randint(0,2,Nbits)
Eb = 1

#Standard: QPSK, 16QAM, 64QAM and 256QAM

if M != 4 and M != 16 and M != 64 and M!= 256:
    print("Invalid modulation. Formats used within 5G-NR mobile
        communications system: QPSK, 16QAM, 64QAM and 256QAM.")
    sys.exit(0)

[BnI,BnQ,AnI,AnQ,AI,AQ] = utils.qam(Bn, Eb, np.sqrt(M),np.sqrt(M))
if (PlotOptions == True):
    plt.figure(figsize=(5,5))
    for ii in range(0,AI.size):
        for jj in range(0,AQ.size):
            plt.plot(AI[ii],AQ[jj], 'ro', label='Constellation')
    plt.xlabel('I'); plt.ylabel('Q');
    plt.grid(True);
    plt.title('Constellation used')
    plt.savefig('../results/txconst.png')
Alphabet = AI
const_points = np.zeros((1,AnI.size),dtype=np.complex_)
const_points.real = AnI
const_points.imag = AnQ
const_points = np.reshape(const_points,(Ncarr_act,NsymbOFDM))
alpha = np.mean(np.abs(const_points))


ofdm_freq = np.zeros((FFTSize,NsymbOFDM));
ofdm_freq = ofdm_freq.astype(np.csingle)
#Adding information
aux = 0
for ii in range(0,NsymbOFDM):
    for jj in range((int)((FFTSize/2)-(Ncarr_act/2)),(int)((FFTSize/2)+(
        Ncarr_act/2))):
        ofdm_freq[jj,ii] = const_points[aux,ii]
        aux = aux + 1
    aux = 0
ofdm_freq = utils.OverSampling_Frequency(ofdm_freq,OvS_freq, True)
if (PlotOptions == True):
    plt.figure()
    plt.stem(np.absolute(ofdm_freq[:,0]), use_line_collection = True)
    plt.xlabel('Subcarriers')
    plt.grid(True)
    plt.title('Subcarriers Tx after Oversampling')
    plt.savefig('../results/subcarrierstx.png')

ofdm_freq = np.fft.fftshift(ofdm_freq)
ofdm_time = np.zeros(ofdm_freq.shape,dtype=np.complex)
for i in range(0,NsymbOFDM):
```

```
        ofdm_time[:,i] = np.fft.ifft(ofdm_freq[:,i])


    def addCP(OFDM_time,NCP):
        cp = OFDM_time[-NCP:]           # take the last CP samples ...
        return np.hstack([cp, OFDM_time]) # ... and add them to the beginning

    x = np.array([])

    #Cyclic Prefix:

    NCP0 = 160/2048*FFTSize *OvS_freq #Overhead - Oversampling done before must
         be taken into account
    NCP = 144/2048*FFTSize *OvS_freq #Overhead 7\%

    for ii in range(0,Nslots*2):
        x = np.hstack([x,addCP(ofdm_time[:,ii*7],(int)(NCP0))])
        for jj in range(7*ii+1,7*ii+7):
            x = np.hstack([x,addCP(ofdm_time[:,jj],(int)(NCP))])
    x = utils.FFTinterpolate(x,Fs,Fo)

    return x, Bn, Alphabet, alpha, Eb, const_points, Bw

def rx_OFDM(Ncarr_act, NsymbOFDM, y, Alphabet, alpha, FFTSize, OvS_freq,
    PlotOptions, Simulation, BwCh, Fs):
        Nsamples = int(FFTSize*OvS_freq) #Nsamples -> We have an oversampling
            that will be fixed in frequency domain
        rx_ofdm_time = np.zeros((Nsamples, NsymbOFDM),dtype=np.complex_)
        y = np.reshape(y,(int(y.size/(int(NsymbOFDM/14*2))),int(NsymbOFDM/14*2))
            ,order='F')
        #Cyclic Prefix:
        NCP0 = int(160/2048*Nsamples) #Overhead
        NCP = int(144/2048*Nsamples) #Overhead 7
        for ii in range(0,int(NsymbOFDM/14*2)):
            rx_ofdm_time[:,ii*7] = y[NCP0:(NCP0+Nsamples),ii]
            for jj in range(1,7):
                rx_ofdm_time[:,jj+ii*7] = y[NCP0+Nsamples+(NCP+Nsamples)*(jj-1)+
                    NCP:NCP0+Nsamples+(NCP+Nsamples)*(jj),ii]
        ACPR, ACPR2 = utils.ACPR_Calculation(rx_ofdm_time, Fs, FFTSize, BwCh,
            OvS_freq)
        print("Adjacent 1st Channel Power Ratio (ACPR): \%.2f dB" \% ACPR)
        if OvS_freq > 5:
            print("Adjacent 2nd Channel Power Ratio (ACPR): \%.2f dB" \% ACPR2)
        rx_ofdm_freq = np.zeros((Nsamples, NsymbOFDM),dtype=np.complex_)
        for i in range(0,NsymbOFDM):
            rx_ofdm_freq[:,i] = np.fft.fft(rx_ofdm_time[:,i])
        rx_ofdm_freq = np.fft.fftshift(rx_ofdm_freq)
        if (PlotOptions == True):
            plt.figure()
            plt.stem(np.absolute(rx_ofdm_freq[:,0]), use_line_collection = True)
            plt.xlabel('Subcarriers')
            plt.grid(True)
            plt.title('Subcarriers Rx')
            plt.savefig('../results/subcarriersrx.png')

        #Downsampling: We want to recover our original signal (Fo MHz instead
            of Fs MHz)
```

```
        rx_ofdm_freq = utils.OverSampling_Frequency(rx_ofdm_freq,OvS_freq,False)
             #False means we want downsampling (remove zeros)
        rx_const_points = np.zeros((Ncarr_act,NsymbOFDM),dtype=np.complex_)
        aux = 0
        for ii in range(0,NsymbOFDM):
            for jj in range((int)((FFTSize/2)-(Ncarr_act/2)),(int)((FFTSize/2)+(
                Ncarr_act/2))):
                rx_const_points[aux,ii] = rx_ofdm_freq[jj,ii]
                aux = aux + 1
            aux = 0
        # From subcarriers to symbols...
        beta = np.mean(np.abs(rx_const_points))
        rx_const_points = rx_const_points * (alpha/beta)
        rx_AnI = rx_const_points.real
        rx_AnQ = rx_const_points.imag
        rx_AnI = rx_AnI.flatten()
        rx_AnQ = rx_AnQ.flatten()
        est_AnI = utils.detectsymb(rx_AnI, Alphabet)
        est_AnQ = utils.detectsymb(rx_AnQ, Alphabet)
        rx_BnI = utils.symb2bit(est_AnI, Alphabet)
        rx_BnQ = utils.symb2bit(est_AnQ, Alphabet)
        k = int(np.log2(Alphabet.size))
        rx_Bn = np.array([],dtype=np.int)
        # ... and now bits (parallel to series conversion included here)
        for ii in range(0,int(rx_BnI.size/k)):
            rx_Bn = np.hstack([rx_Bn,rx_BnI[ii*k:ii*k+k]])
            rx_Bn = np.hstack([rx_Bn,rx_BnQ[ii*k:ii*k+k]])
        return rx_Bn, rx_const_points, ACPR, ACPR2

def Channel(x, SNRdb, Chtype):
    """
    Description
    ----------
    Channel model of the simulated environment

    Parameters
    ----------
    x : numpy of array complex
        input signal.
    SNRdb : float
        signal to noise ratio specified in decibels.
    Chtype : string
        Channel type.

    Returns
    -------
    y : numpy of array complex
        output signal of the channel stage.

    """
    if (Chtype == "AWGN"):
        nsr = 10**(-SNRdb/10)
        Ps = np.sum(np.abs(x**2))/x.size
        n = np.random.randn(len(x)*2).view(np.complex128)/np.sqrt(2) #Noise
            with power 1
        n = np.sqrt(Ps*nsr)*n #We set noise power to meet SNR specifications
        y = x + n
```

```
    return y

def MulticarrierReceiver(NRB, MyStructure, y, Nslots, Alphabet, alpha, FFTSize,
     Fs, Fo, PlotOptions, Simulation, BwCh, Filtering):
    Ncarr_act = NRB * 12
    NsymbOFDM = Nslots*14
    OvS = Fs / Fo
    OvS_freq = np.floor(OvS)
    f, Pyy = utils.PSD(y, Fs, FFTSize)
    Pyy  = 10*np.log10(Pyy) + 30
    if (PlotOptions == True):
        plt.figure(figsize=(5, 4))
        plt.plot(f,Pyy)
        plt.xlabel('frequency [MHz]')
        plt.ylabel('PSD [dBm/Hz]')
        plt.title('Power Spectral Density of the signal received')
        plt.show()
        plt.grid(True);
        plt.savefig('../results/PSDrx.png')

    #Interpolation in time domain (Downsampling)
    y = utils.FFTinterpolate(y,Fo*OvS_freq,Fs)

    rx_Bn, rx_const_points, ACPR, ACPR2 = rx_OFDM(int(Ncarr_act), int(NsymbOFDM
        ), y, Alphabet, alpha,FFTSize, OvS_freq, PlotOptions, Simulation, BwCh,
         Fs)
    return rx_Bn, rx_const_points, ACPR, ACPR2
```

A.3 Some other util functions

Code A.3 Some other util functions (`NRutils.py`).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Author: Guillermo Palomino Lozano
Organization: University of Seville
Date: May 2020
"""
import sys
import os
import numpy as np
import matplotlib.pyplot as plt
from math import pi
from scipy.signal import kaiserord, firwin, freqz, welch
from scipy.special import erfc
from sympy.combinatorics.graycode import GrayCode


def PSD(y, Fs, FFTSize):
    """
    Description
    -----------
    Compute the power spectral density for a given signal with a frequency
        sampling Fs and using FFTSize points

    Parameters
    ----------
    y : numpy complex array
        input signal.
    Fs : float
        frecuency sampling.
    FFTSize : int
        number of points required for the computing of the fast fourier
            transform algorithm.

    Returns
    -------
    f : numpy float array
        frequency grid.
    Pxx : numpy complex array
        power spectral density of the given signal y.

    """

    f_aux, Pxx_aux = welch(y, Fs, nperseg=FFTSize)
    f = np.array([])
    Pxx = np.array([])
    length = len(f_aux)
    f=np.concatenate((f,f_aux[int(FFTSize/2):length]))
    f=np.concatenate((f,f_aux[0:int(FFTSize/2)]))
    f = f/1e6
    Pxx=np.concatenate((Pxx,Pxx_aux[int(FFTSize/2):length]))
```

```
    Pxx=np.concatenate((Pxx,Pxx_aux[0:int(FFTSize/2)]))
    return f, Pxx

def ACPR_Calculation(y, Fs, FFTSize, BwCh, OvS):
    """
    Description
    ----------
    Compute the adjacent channel power ratio (ACPR) for a given signal y

    Parameters
    ----------
    y : numpy complex array
        input signal.
    Fs : float
        frecuency sampling.
    FFTSize : int
        number of points required for the computing of the fast fourier
            transform algorithm.
    BwCh : int
        Channel bandwidth.
    OvS : float
        Oversampling needed for meet the requirement of the frequency sampling
            desired.

    Returns
    -------
    ACPR : float
        ACPR of the first adjacent channel.
    ACPR2 : float
        ACPR of the second adjacent channel.

    """

    Nsamples, Nsymb = y.shape
    Pchann = np.zeros(Nsymb)
    Pabov = np.zeros(Nsymb)
    Pbelw = np.zeros(Nsymb)

    Pabov2 = np.zeros(Nsymb)
    Pbelw2 = np.zeros(Nsymb)
    for ii in range(Nsymb):
        yaux = y[:,ii]

        f, psdy = PSD(yaux, Fs, FFTSize)

        idx_chann = (f > -(BwCh/2)) * (f < (BwCh/2))
        idx_abov = (f > (BwCh/2)) * (f < ((BwCh/2) + BwCh))
        idx_belw = (f < (-BwCh/2)) * (f > ((-BwCh/2) - BwCh))

        Pchann[ii] = np.sum(psdy[idx_chann])*BwCh*1e6
        Pabov[ii] = np.sum(psdy[idx_abov])*BwCh*1e6
        Pbelw[ii] = np.sum(psdy[idx_belw])*BwCh*1e6

        if OvS > 5:
            #Also the second adjacent channel

            idx_abov2 = (f > (BwCh/2 + BwCh)) * (f < ((BwCh/2) + 2*BwCh))
```

```
            idx_belw2 = (f < (-BwCh/2) - BwCh) * (f > ((-BwCh/2) - 2*BwCh))

            Pabov2[ii] = np.sum(psdy[idx_abov2])*BwCh
            Pbelw2[ii] = np.sum(psdy[idx_belw2])*BwCh


    ACPR = 10*np.log10(np.mean(np.array([np.mean(Pabov),np.mean(Pbelw)]))/np.
        mean(Pchann))
    ACPR2 = False
    if OvS > 5:
        ACPR2 = 10*np.log10(np.mean(np.array([np.mean(Pabov2),np.mean(Pbelw2)]))
            /np.mean(Pchann))
    return ACPR, ACPR2
def EVM_Calculation(const_points, rx_const_points):
    """
    Description
    -----------
    Compute the error vector magnitude (EVM) between the original and resulting
         point of the constellation

    Parameters
    ----------
    const_points : numpy array complex
        constellation points transmitted.
    rx_const_points : numpy array complex
        constellation points received.

    Returns
    -------
    EVM : float
        EVM performance.

    """
    EVM_Reference = np.amax(np.sqrt(const_points.real**2+const_points.imag**2))
    err = const_points - rx_const_points;
    EVM = np.sqrt(1/err.size*np.sum(err.real**2 + err.imag**2))/EVM_Reference *
         100
    return EVM

def NMSE_Calculation(x,y):
    """
    Description
    ----------
    Compute the normalized mean square error for to given values x and y

    Parameters
    ----------
    x : numpy array of complex
        signal 1.
    y : numpy array of complex
        sigtnal 2.

    Returns
    -------
    float
        NMSE.
```

```
    """
    return (np.sum(np.abs((x - y))**2))/(np.sum(np.abs(x)**2))

def DrawConstellation(const_points, rx_const_points, NUMPOINTS):
    """
    Description
    ----------
    Draw the constellation with tx and rx points, for a given number of points
        NUMPOINTS

    Parameters
    ----------
    const_points : numpy array complex
        constellation points transmitted.
    rx_const_points : numpy array complex
        constellation points received.
    NUMPOINTS : int
        Number of points desired for plotting the constellation.

    Returns
    -------
    None.

    """
    plt.figure(figsize=(5,5))
    index = np.random.randint(0, high=len(rx_const_points), size=NUMPOINTS)
    for ii in range(len(index)):
        plt.plot(rx_const_points.real[ii],rx_const_points.imag[ii],'-o',color='
            blue')
        plt.plot(const_points.real[ii],const_points.imag[ii],'-o',color='red')
    plt.xlabel('I'); plt.ylabel('Q');
    plt.grid(True);
    plt.title('Constellation received')
    plt.savefig('../results/constellationrx.png')

def OverSampling_Frequency(X,OvS,Upsampling):
    """
    Description
    ----------
    Perform the oversampling in frequency domain by adding/removing zeros

    Parameters
    ----------
    X : numpy array complex
        input signal.
    OvS : float
        Oversampling factor.
    Upsampling : boolean
        True if upsampling must be performed, false if downsampling.

    Returns
    -------
    Y : numpy array complex
        input signal after oversampling.

    """
    #Zero padding OvS times the length of X -> we change fs
```

```
    if (OvS != 1):
        (Nx,Nsymb) = np.shape(X)
        if (Upsampling == True):
            #Upsampling: Add zeros
            Ny = np.round(Nx*OvS)
            Y = np.zeros((int(Ny),Nsymb),dtype=np.complex_)
            if Nx \% 2 == 0:
                Y.real[int(Ny/2 - np.floor(Nx/2)):int(Ny/2 + np.floor(Nx/2)),:]
                    = X.real
                Y.imag[int(Ny/2 - np.floor(Nx/2)):int(Ny/2 + np.floor(Nx/2)),:]
                    = X.imag
            else:
                Y.real[int(Ny/2 - np.floor(Nx/2)):int(Ny/2 + np.floor(Nx/2)+1)
                    ,:] = X.real
                Y.imag[int(Ny/2 - np.floor(Nx/2)):int(Ny/2 + np.floor(Nx/2)+1)
                    ,:] = X.imag
        else:
            #Downsampling: Remove zeros
            Ny = np.round(Nx/OvS)
            Y = np.zeros((int(Ny),Nsymb),dtype=np.complex_)
            if Nx \% 2 == 0:
                Y.real = X.real[int(Nx/2 - np.floor(Ny/2)):int(Nx/2 + np.floor(
                    Ny/2)),:]
                Y.imag = X.imag[int(Nx/2 - np.floor(Ny/2)):int(Nx/2 + np.floor(
                    Ny/2)),:]
            else:
                Y.real = X.real[int(Nx/2 - np.floor(Ny/2)):int(Nx/2 + np.floor(
                    Ny/2)+1),:]
                Y.imag = X.imag[int(Nx/2 - np.floor(Ny/2)):int(Nx/2 + np.floor(
                    Ny/2)+1),:]
    else:
        Y = X
    return Y

def RootRaisedCos(Nfilt,B=0,T=1):
    """
    Description
    ----------
    Define root raised cosine function

    Parameters
    ----------
    Nfilt : int
        Filter length.
    B : float, optional
        Roll-off factor of root raised cosine function. The default is 0.
    T : float, optional
        Reciprocal of the symbol-rate. The default is 1.

    Returns
    -------
    TYPE
        DESCRIPTION.

    """
    x = np.linspace(-Nfilt/2, Nfilt/2, Nfilt+1)
```

```
    return (1/np.sqrt(T))*(np.sin(pi*x/T*(1-B))+4*B*x/T*np.cos(pi*x/T*(1+B)))/(
        pi*x/T*(1-(4*B*x/T)**2))

def primeFactors(n):
    """
    Description
    ----------
    Compute the prime factors of a number n

    Parameters
    ----------
    n : int
        input number.

    Returns
    -------
    result : array of float
        prime factors of number n.

    """

    result = np.array([],dtype=np.int)
    # Print the number of two's that divide n
    while n \% 2 == 0:
        result = np.append(result,2)
        n = n / 2
    # n must be odd at this point
    # so a skip of 2 ( i = i + 2) can be used
    for i in range(3,int(np.sqrt(n))+1,2):

        # while i divides n , print i ad divide n
        while n \% i== 0:
            result = np.append(result,i)
            n = n / i

    # Condition if n is a prime
    # number greater than 2
    if n > 2:
       result = np.append(result,n)

    return result

def resample_quotients(fs1, fs2):
    """
    Description
    ----------
    Compute the P and Q resampling coefficients to be used in FFTinterpolate

    Parameters
    ----------
    fs1 : int
        frequency sampling 1.
    fs2 : int
        frequency sampling 2.

    Returns
    -------
```

```
    P : float
        P coefficient.
    Q : float
        Q coefficient.

    """
    v1 = primeFactors(fs1)
    v2 = primeFactors(fs2)
    total_ind = np.array([])
    for k in range(0,v1.size):
        #If we can find element k of v1 in v2
        if np.isin(v1[k],v2):
            #Find first index in v2 where it can be found
            ind = np.nonzero(v1[k] == v2)[0][0]
            #Remove the value at index k from v1
            total_ind = np.append(total_ind,k)
            #Remove the value at index ind from v2
            v2 = np.delete(v2,ind)

    P = np.prod(v1[np.setdiff1d(np.arange(0,v1.size),total_ind)])
    Q = np.prod(v2)
    return P,Q

def FFTinterpolate(x,fs_y,fs_u):
    """
    Description
    ----------
    Interpolation performed by FFt algorithm

    Parameters
    ----------
    x : numpy array of complex
        input signal.
    fs_y : float
        sampling rate of the output signal.
    fs_u : TYPE
        sampling rate of the input signal.

    Returns
    -------
    y : TYPE
        DESCRIPTION.

    """
    if (fs_u != fs_y):
        N = x.size
        P, Q = resample_quotients(fs_y, fs_u)
        Nn = float(N*P/Q)
        U = np.fft.fft(x)
        Y = np.zeros((int(Nn)),dtype=np.complex_)
        if (np.round(Nn) == Nn):
            Nn = int(Nn)
            Y[Nn-1] = (1j)*10**(-16)
            if (P > Q):
                #Upsampling
                if (Nn \% 2) == 0:
                    if (N \% 2) == 0:
```

```
                        #Even number of samples in u
                        #Easy to put back
                        N = int(N)
                        Y[0:int(N/2)] = U[0:int(N/2)]
                        Y[Nn-int(N/2):Nn] = U[int(N/2):N]
                    else:
                        Y[0:int(np.floor(N/2))] = U[0:int(np.floor(N/2))]
                        Y[Nn-int(np.ceil(N/2)):Nn] = U[int(np.floor(N/2)):N]
                else:
                    print("Not implemented")
                    sys.exit(0)
            else:
                #Downsampling
                Y[0:int(np.ceil(Nn/2))] = U[0:int(np.ceil(Nn/2))]
                Y[Nn-int(np.ceil(Nn/2)):Nn] = U[N-int(np.ceil(Nn/2)):N]
        else:
            print("Not an integer number of samples. Use some other method")
            sys.exit(0)
        y = np.fft.ifft(Y)
    else:
        y = x
    return y

def Filter_signal(Fs, BwCh, Bw, x):
    """
    Description
    ----------
    Function that filter an input signal using the Kaiser window method.
    Parameters
    ----------
    Fs : float
        frequency sampling.
    BwCh : float
        Channel bandwidth available.
    Bw : float
        bandwidth of the input signal.
    x : numpy array of complex
        input signal.

    Returns
    -------
    y : numpy array of complex
        signal filtered.

    """
    # The Nyquist rate of the signal.
    nyq_freq = Fs / 2.0

    Fend = (BwCh / 2) / nyq_freq
    # The cutoff frequency of the filter.
    Fc = (Bw*1.025 / 2) / nyq_freq

    # The desired width of the transition from pass to stop,
    # relative to the Nyquist rate. We'll design the filter
    # with a 5 Hz transition width.
    width = Fend - Fc
    # The desired attenuation in the stop band, in dB.
```

```
    ripple_db = 100

    # Compute the order and Kaiser parameter for the FIR filter.
    N, beta = kaiserord(ripple_db, width)
    if (N \% 2 == 0):
        N = N+1
    # Use firwin with a Kaiser window to create a lowpass FIR filter.
    taps = firwin(N, Fc , window=('kaiser', beta))
    delay = int(0.5 * (N))
    # Use lfilter to filter x with the FIR filter.
    #yz = lfilter(taps, 1.0, xz)
    #y = yz[delay:]
    yaux = np.fft.ifft(np.fft.fft(x)*np.fft.fft(taps,x.size))
    y = np.array([])
    y = np.append(y,yaux[delay:])
    y = np.append(y,yaux[0:delay])
    #-----------------------------------------------
    # Plot the magnitude response of the filter.
    #-----------------------------------------------

    plt.figure(2)
    plt.clf()
    w, h = freqz(taps, worN=8000)
    plt.plot((w/pi)*nyq_freq/1e6, 10*np.log10(np.abs(h)), linewidth=2, label="H
        (f)")
    f, Pxx = PSD(x, Fs, 8000)
    Pxx  = 10*np.log10(Pxx) + 30
    f2, Pyy = PSD(yaux, Fs, 8000)
    Pyy  = 10*np.log10(Pyy) + 30
    plt.plot(f,Pxx - np.max(Pxx),label="X(f)")
    plt.plot(f2,Pyy- np.max(Pyy),label="Y(f)")
    plt.xlabel('Frequency (MHz)')
    plt.ylabel('Gain')
    plt.title('Frequency Response')
    plt.legend(loc="upper right")
    plt.grid(True)
    plt.xlabel('frequency [MHz]')
    plt.ylabel('PSD [dBm/Hz]')
    plt.show()
    plt.grid(True);
    plt.title('Filter effect')
    plt.grid(True)
    plt.savefig('../results/filteringx.png')
    return y

def split(Bn, M1, M2):
    k1=int(np.log2(M1))
    k2=int(np.log2(M2))
    k=k1+k2
    Nb=len(Bn)
    W=np.reshape(Bn,[int(Nb/k),k])
    BnI=np.reshape(W[:,:k1],[k1*int(Nb/k)])
    BnQ=np.reshape(W[:,k1:],[k2*int(Nb/k)])
    return BnI,BnQ

def de2gray(d,n):
```

```
    gray_list_str = list(GrayCode(n).generate_gray())
    gray_list = list(map(lambda ind: np.array(list(gray_list_str[ind]), dtype=
        np.int), range(len(gray_list_str))))
    g = list(map(lambda ind: gray_list[int(d[ind])], range(0,len(d))))

    return np.array(g)

def detectsymb(rn, alfabeto):

    N = len(rn)

    An = np.zeros(N)

    for i in range(N):
        ind = np.where(abs(rn[i]-alfabeto) == np.amin(abs(rn[i]-alfabeto)))
        An[i] = alfabeto[ind]

    return An

def gray2de(b):
    """
    Description
    ----------
    Gray to decimal converter

    Parameters
    ----------
    b : numpy array of int
        input bits.

    Returns
    -------
    d : numpy array of int
        output bits.

    """
    from numpy import zeros_like, logical_xor, fliplr, shape, reshape, matrix,
        arange, array
    b=matrix(b)
    c=zeros_like(b)
    c[:,0] = b[:,0];

    for i in range(1,shape(b)[1]):
        c[:,i] = logical_xor(c[:,i-1], b[:,i])

    c=fliplr(c);

    [n,m] = shape(c)
    if min([m,n]) < 1:
        d = [];
        return
    elif min([n,m]) == 1:
        m = max([n,m])
        n = 1
        c = reshape(c,[n,m])
```

```
    d = array((c * matrix(2**arange(m)).T).T).squeeze();

    return d

def Qfunct(x):
    """
    Calculation of Q function for BER performance
    ----------

    Parameters
    ----------
    x : numpy array of complex
        input signal.

    Returns
    -------
    res : float
        Q(x).

    """
    res=(1/2)*erfc(x/np.sqrt(2))
    return res

def symb2bit(An,Alphabet):
    """
    Description
    ----------
    Symbol to bit converter

    Parameters
    ----------
    An : numpy array of float
        array of symbols.
    Alphabet : numpy array of float
         alphabet of the QAM modulation.

    Returns
    -------
    Bn : numpy array of int
        sequence of bits corresponding to the input symbols.

    """
    k = np.log2(len(Alphabet))

    if k>1:
        dist = abs(Alphabet[0]-Alphabet[1])
        idx = np.round((An-Alphabet[0])/dist)
        Bn = np.reshape(de2gray(idx,k),[int(k*len(An))])
    else:
        Bn = ((An/max(Alphabet))+1)/2

    return Bn



def qam(Bn, Eb, M1, M2):
    """
```

```
    Description
    -----------
    QAM modulator

    Parameters
    ----------
    Bn : numpy array of int
        sequence of bits to transmit.
    Eb : float
        Energy per bit.
    M1 : int
        Modulation order 1.
    M2 : int
        Modulation order 2.

    Returns
    -------
    BnI : numpy array of int
        bits sequence of in-phase component.
    BnQ : numpy array of int
        bit sequence of quadrature component.
    AnI : numpy array of float
        transmitted levels in in-phase component.
    AnQ : numpy array of float
        transmitted levels in quadrature component.
    AI : numpy array of float
        used levels used in in-phase component.
    AQ : numpy array of float
        used levels in quadrature component.

    """

    k1=int(np.ceil(np.log2(M1)))
    M1=2**(k1)
    k2=int(np.ceil(np.log2(M2)))
    M2=2**(k2)

    k=k1+k2
    Nb=len(Bn)
    Bn=np.r_[Bn, np.zeros(int(k*np.ceil(Nb/k)-Nb), dtype=int)]

    A= np.sqrt(3*Eb*np.log2(M1*M2)/(M1**2+M2**2-2))

    AI=A*(2*np.arange(M1)-M1+1)
    AQ=A*(2*np.arange(M2)-M2+1)

    BnI,BnQ=split(Bn,M1,M2)

    NbI=len(BnI)
    NbQ=len(BnQ)

    if M1>2:
        AnI=AI[gray2de(np.reshape(BnI,[int(NbI/k1),k1]))]
    else:
        AnI=AI[BnI]
```

```
    if M2>2:
        AnQ=AQ[gray2de(np.reshape(BnQ,[int(NbQ/k2),k2]))]
    else:
        AnQ=AQ[BnQ]

    return BnI,BnQ,AnI,AnQ,AI,AQ
```

A.4 Extra code for some figures generation

Code A.4 Script that generates Fig.4 4.2 and Fig. 4.3.

```
# -*- coding: utf-8 -*-
"""
Author: Guillermo Palomino Lozano
Organization: University of Seville
Date: May 2020
"""
import numpy as np
import os
import matplotlib.pyplot as plt
import scipy.io as sio
import tikzplotlib
import mc5g as NR
import ccdd as ccdd
import matlab.engine
from matplotlib.ticker import ScalarFormatter
import warnings
#------------------------------------------------------------
plt.close('all')

NRB = 250 # Number of resource blocks
Df = 15e3 # Subcarrier Spacing
OvS = True # Oversampling
Fs = 100e6 # Only important when oversampling is true
Nslots = 1
FR = 1 # Frecuency range
PAPR = 10.5 #PAPR desiredPlotOptions = False #True for active plots (
    Subcarriers Tx/Rx, Constellation, Spectrum, etc )
#-----------------------------------------------Pl-------------
#   Simulations available:
Simulation = False
#------------------------------------------------------------
warnings.filterwarnings('ignore')
NUMSIM = 1000
PAPRx = np.zeros(NUMSIM)
PAPRy = np.zeros(NUMSIM)
Nsamples = np.zeros(NUMSIM)
M = np.array([16])
k = np.log2(M)
MyStructure = NR.Numerology(Df,FR)
print("\n------------- STARTING TX -------------")
if Simulation == True:
    for jj in range(0,1000):
        SEED = jj
        print("Simulation number %d" % jj)
        x, Bn, FFTSize, Alphabet, alpha, Eb, const_points, Fs, Fo, BwCh, PAPRx[
            jj], PAPRy[jj], Nsamples[jj] = NR.MulticarrierGeneration(MyStructure
            , NRB, OvS, Fs, Nslots, M, PAPR, SEED, False, False)
        np.savez('PAPR.npz',PAPRx=PAPRx,PAPRy=PAPRy, Nsamples=Nsamples)
else:
    data = np.load('PAPR.npz')
    PAPRx = data['PAPRx']
```

```
    PAPRy = data['PAPRy']
plt.figure()
n, bins, patches = plt.hist(x=PAPRx, bins=30, color='#607c8e',alpha=0.7, rwidth
    =0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('PAPR (dB)')
plt.ylabel('Counts')
tikzplotlib.save("Hist.tex")

plt.figure()
plt.plot(np.arange(0,1000),PAPRy, label='Output')
plt.plot(np.arange(0,1000),PAPRx, label='Input')
plt.grid()
plt.xlabel('Simulations')
plt.ylabel('PAPR (dB)')
plt.legend()
tikzplotlib.save("PAPRalong.tex")


error = np.mean(np.abs(10.5 - PAPRy)) * 100
print("PAPR error after peak clipping: %.5f" % (error))
print("Average of samples affected: %.2f " % (np.mean(Nsamples)))
```

Code A.5 Script that generates Fig. 4.7 and Fig. 4.6.

```
# -*- coding: utf-8 -*-
"""
Author: Guillermo Palomino Lozano
Organization: University of Seville
Date: May 2020
"""

import numpy as np
import matplotlib.pyplot as plt
import tikzplotlib
plt.close('all')
data = np.load('040620PowAmp2.npz')
Xamp = data['X_amp']
Yamp = data['Y_amp']
Pin = data['Pin']
XdB = 10*np.log10(np.abs(Xamp)**2/50)+30
YdB = 10*np.log10(np.abs(Yamp)**2/50)+30
G = YdB - XdB
leg=["-30dBm","-29dBm","-28dBm","-27dBm","-26dBm","-25dBm","-24dBm","-23dBm
    ","-22dBm","-21dBm"]
plt.figure()
for ii in range(len(Pin)):
    plt.scatter(XdB[len(Pin) - ii - 1,:], G[len(Pin) - ii - 1,:],s=0.5, label=
        leg[len(Pin) - ii - 1])
plt.xlabel('Input Power')
plt.ylabel('Gain')
plt.title('Gain / AM')
plt.xlim(-50, -8)
plt.ylim(25, 65)
plt.figure()
for ii in range(len(Pin)):
    plt.scatter(XdB[len(Pin) - ii - 1,:], YdB[len(Pin) - ii - 1,:],s=0.5,label=
        leg[len(Pin) - ii - 1])
plt.xlabel('Input Power')
plt.ylabel('Output Power')
plt.title('AM / AM')
plt.ylim((-30, 65))
```

List of Figures

List of Tables

List of Codes

Bibliography

[ETS, 2018] (2018). *Physical channels and modulation, 3GPP TS 38.211 version 15.2.0 Release 15*.

[Cis, 2020] (2020). Cisco annual internet report (2018–2023). Technical report, White paper, Cisco public.

[ROA, 2020] (2020). *Release timeline 5G*. 3GPP, ETSI.

[Ari and Ustazhanov, 2014] Ari, N. and Ustazhanov, M. (2014). Matplotlib in Python. *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–6.

[Cheng et al., 2016] Cheng, X., He, Y., Ge, B., and He, C. (2016). A filtered OFDM using FIR filter based on window function method. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*. IEEE.

[Dahlman et al., 2018] Dahlman, E., Parkvall, S., and Sköld, J. (2018). Overall transmission structure. In *5G NR: the Next Generation Wireless Access Technology*, pages 103–131. Elsevier.

[Fangohr, 2004] Fangohr, H. (2004). A comparison of C, MATLAB, and Python as teaching languages in engineering. In *Computational Science - ICCS 2004*, pages 1210–1217. Springer Berlin Heidelberg.

[Ghosh et al., 2019] Ghosh, A., Maeder, A., Baker, M., and Chandramouli, D. (2019). 5G evolution: A view on 5G cellular technology beyond 3GPP release 15. *IEEE Access*, 7:127639–127651.

[GMB, 2006] GMB, N. R. (2006). Technology of high speed packet access (hspa).

[Gomes et al., 2018] Gomes, R., Sismeiro, L., Ribeiro, C., Fernandes, T. R., Sanchez, M. G., Hammoudeh, A., and Caldeirinha, R. F. S. (2018). Will COTS RF front-ends really cope with 5G requirements at mmWave? *IEEE Access*, 6:38745–38769.

[Guan et al., 2017] Guan, P., Wu, D., Tian, T., Zhou, J., Zhang, X., Gu, L., Benjebbour, A., Iwabuchi, M., and Kishiyama, Y. (2017). 5G field trials: OFDM-based waveforms and mixed numerologies. *IEEE J. Sel. Areas Commun.*, 35(6):1234–1243.

[Gupta and Jha, 2015] Gupta, A. and Jha, R. K. (2015). A survey of 5G network: Architecture and emerging technologies. *IEEE Access*, 3:1206–1232.

[Henry et al., 2020] Henry, S., Alsohaily, A., and Sousa, E. S. (2020). 5G is real: Evaluating the compliance of the 3GPP 5G new radio system with the ITU IMT-2020 requirements. *IEEE Access*, 8:42828–42840.

[Ijaz et al., 2016] Ijaz, A., Zhang, L., Grau, M., Mohamed, A., Vural, S., Quddus, A. U., Imran, M. A., Foh, C. H., and Tafazolli, R. (2016). Enabling massive IoT in 5G and beyond systems: PHY radio frame design considerations. *IEEE Access*, 4:3322–3339.

[ITU-R, 2015] ITU-R (2015). *Framework and overall objectives of the future development of IMT2020 for 2020 and beyond. Recommendation ITU-R M.2083*.

[Kaltenberger et al., 2019] Kaltenberger, F., d. Souza, G., Knopp, R., and Wang, H. (2019). The openairinterface 5G new radio implementation: Current status and roadmap. In *WSA 2019; 23rd International ITG Workshop on Smart Antennas*, pages 1–5.

[Landin et al., 2015] Landin, P. N., Gustafsson, S., Fager, C., and Eriksson, T. (2015). WebLab: A Web-Based Setup for PA Digital Predistortion and Characterization [Application Notes]. *IEEE Microw. Mag.*, 16(1):138–140.

[Levanen et al., 2017] Levanen, T., Kaikkonen, J., Nielsen, S., Pajukoski, K., Renfors, M., and Valkama, M. (2017). 5G new radio UL coverage with peak clipping. In *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE.

[Lin et al., 2017] Lin, P.-C., Huang, S.-L., and Li, X.-Y. (2017). Teaching and learning next generation mobile communication networks through open source openAirInterface testbeds. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE.

[López, 2020] López, D. (2020). Técnicas de reducción del factor de cresta aplicadas a señales 5g new radio. Master's thesis, Universidad de Sevilla.

[Marcus, 2015] Marcus, M. J. (2015). 5G and "IMT for 2020 and beyond" [Spectrum Policy and Regulatory Issues]. *IEEE Wireless Commun.*, 22(4):2–3.

[Ochiai and Imai, 2001] Ochiai, H. and Imai, H. (2001). On the distribution of the peak-to-average power ratio in OFDM signals. *IEEE Trans. Commun.*, 49:282 – 289.

[Oliphant, 2004] Oliphant, T. E. (2004). Scipy tutorial.

[Oliphant, 2007] Oliphant, T. E. (2007). Python for scientific computing. *Comput. Sci. Eng.*, 9(3):10–20.

[Oppenheim et al., 2001] Oppenheim, A. V., Buck, J. R., and Schafer, R. W. (2001). *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall.

[Osseiran et al., 2014] Osseiran, A., Boccardi, F., Braun, V., Kusume, K., Marsch, P., Maternia, M., Queseth, O., Schellmann, M., Schotten, H., Taoka, H., Tullberg, H., Uusitalo, M. A., Timus, B., and Fallgren, M. (2014). Scenarios for 5G mobile and wireless communications: the vision of the METIS project. *IEEE Commun. Mag.*, 52(5):26–35.

[Ozgur et al., 2017] Ozgur, C., Colliau, T., Rogers, G., Hughes, Z., and Bennie, E. (2017). MatLab vs. Python vs. R. *Journal of Data Science: JDS*, 15:355–372.

[Palomino et al., 2020] Palomino, G., Pérez-Hernández, A., Madero-Ayora, M. J., and Becerra, J. A. (2020). An open source project for learning 5G waveforms.

[Park et al., 2019] Park, J., Lee, E., Park, S.-H., Raymond, S.-S., Pyo, S., and Jo, H.-S. (2019). Modeling and analysis on radio interference of OFDM waveforms for coexistence study. *IEEE Access*, 7:35132–35147.

[Proakis and Salehi, 2008] Proakis, J. and Salehi, M. (2008). *Digital Communications*. McGraw-Hill.

[Rateb and Labana, 2019] Rateb, A. M. and Labana, M. (2019). An optimal low complexity PAPR reduction technique for next generation OFDM systems. *IEEE Access*, 7:16406–16420.

[Remmert, 2019] Remmert, H. (2019). What is 5G? part 1 - evolution and the next generation.

[Salama and Elmesalawy, 2019] Salama, A. I. and Elmesalawy, M. M. (2019). Flexible and adaptive testbed for 5G experimentations. In *2019 Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE.

[Selva, 2015] Selva, J. (2015). FFT Interpolation from nonuniform samples lying in a regular grid. *IEEE Trans. Signal Process.*, 63(11):2826–2834.

[Truong et al., 2014] Truong, T.-A., Arzel, M., Lin, H., Jahan, B., and Jézéquel, M. (2014). Dft precoded ofdm—an alternative candidate for next generation pons. *J. Lightwave Technol.*, 32(6):1228–1238.

[van der Walt et al., 2011] van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, 13:22–30.

[Zaidi, 2017] Zaidi, A. (2017). Three design principles of 5G New Radio.