Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Simulación de sistemas de radiocomunicación mediante Python

Autor: María Teresa Jurado Cañero
Tutor: María José Madero Ayora

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

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Simulación de sistemas de radiocomunicación mediante Python

Autor:
María Teresa Jurado Cañero

Tutor:
María José Madero Ayora
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Simulación de sistemas de radiocomunicación mediante Python

Autor: María Teresa Jurado Cañero

Tutor: María José Madero Ayora

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos y compañeros

Agradecimientos

Estos cuatro años han sido muy intensos y duros, no obstante, hoy puedo escribir esto y es gracias a todas las personas que me han apoyado y motivado para seguir siempre adelante. En primer lugar, le agradezco a mis padres por haberme dado ánimos en todo momento y ayudarme tanto económica como emocionalmente. En segundo lugar, a mis amigos y compañeros de la universidad, porque me han enseñado que hay algo más a parte de lo que es estudiar un examen y es el trabajar en equipo, confiar en otras personas y a alegrarse por los éxitos de los demás, aunque a ti no te saliesen las cosas bien. A mis compañeros de residencia, sobre todo a Paloma y a mis compañeras de piso Nuria, María, Laura y Montse, por aguantar la presión a la que estaba sometida siempre y por convertirse en mi familia sevillana. A mis amigos de Córdoba, por todos esos veranos dándome ánimo mientras todo el mundo estaba de vacaciones y por la forma en la que siempre han confiado en mí más que yo misma.

A lo largo de estos años, he tenido profesores de todo tipo, más allá de sus formas de explicar que nos pueden gustar más o menos, agradezco a todos esos profesores a los que les gusta su trabajo y disfrutan haciéndolo, ya que son los que más me han enseñado. Sobre todo, a María José, por ayudarme tanto con este trabajo que me ha dado infinitos problemas.

En definitiva, le agradezco a todo el mundo que se ha cruzado en mi camino a lo largo de estos años ya que todos me han enseñado algo.

María Teresa Jurado Cañero

Sevilla, 2019

Resumen

En este trabajo de fin de grado se realizará la simulación de un sistema de radiocomunicación mediante Python de forma que se estudie la tasa de error de bit. Dada la importancia de la BER en un sistema, se tratará de minimizarla implementando varias técnicas de modulación como: QAM, PSK, OFDM/QAM y OFDM/PSK. El diseño se realizará de forma práctico-teórica y la implementación se hará con la ayuda de librerías de software libre de Python especializadas en aspectos de comunicaciones digitales.

Abstract

The aim of this Project is the simulation of a RF system using Python as the code lenguaje in order to study the bit error rate. Due to the importance of the BER in a system, the minimization of it will be research using a range of modulation technices such as: QAM, PSK, OFDM/QAM and OFDM/PSK. The design will be done in a practical and theoretical way and the implementation will be developed regarding the free-software libraries of Python.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

F_x	Función distribución de probabilidad
$P(X \leq x)$	Probabilidad de que ocurra un suceso
μ_x	Media
σ_x	Varianza
$E\{g(x)\}$	Valor esperado
f_x	Función densidad de probabilidad
dx	Derivada de x
R_x	Autocorrelación de x
$R_{x,y}$	Correlación de x e y
τ	Tau
S_X	Densidad espectral de potencia
N_0	Ruido Gaussiano
$\propto$	Factor de *roll off*
cos	Función coseno
sin	Función coseno
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
SNR	Relación señal a ruido
Eb/No	Relación Energía por bit a ruido
BER	Tasa de error de bit
Pb	Probabilidad de error de bit
$\phi(t)$	Pulso ortogonal
e	Numero e
PSK	Modulación en fase
QAM	Modulación en cuadratura
OFDM	Multiplexación por división ortogonal en frecuencia
M	Número de símbolos
FFT	Transformada rápida de Fourier
IFFT	Transformada inversa de Fourier
ISI	Interferencia entre símbolos
CP	Prefijo cíclico
Q(x)	Función Q

1 INTRODUCCIÓN

El mundo está rodeado de constantes transmisiones ya sea por ocio o por trabajo, por lo que la importancia de que los sistemas estén correctamente implementados es necesaria para cualquier actividad en nuestro día a día. Cualquier ingeniero o ingeniera de telecomunicaciones conoce de forma teórica cómo funciona un sistema de comunicaciones digitales, sin embargo, con este proyecto he querido acercarme más al mundo real sobre cómo implementar un sistema que nos permita mandar y recibir información y evaluar su calidad mediante la tasa de error de bit ya que al fin y al cabo la funcionalidad de las telecomunicaciones es poder entablar una comunicación tomando como símil la conversación entre dos personas.

Hay muchas formas de implementar un sistema de comunicaciones digitales cambiando distintos bloques en su estructura y diseñándolos de distintas formas. En este trabajo de fin de grado se estudiará la tasa de error de bit o BER de un sistema de comunicaciones digitales con el fin de que la información llegue de un transmisor a un receptor con la mínima probabilidad de error posible, comparándolo con las características y costes que tendría este. Se implementarán distintas técnicas de modulación para obtener las mejores opciones a la hora de transmitir información. No es un tema novedoso ya que se puede encontrar mucha información sobre la simulación de un sistema de radiocomunicación en Matlab, sin embargo, sí lo es que se implemente en Python que es un lenguaje de programación libre, utilizando librerías de código libre también, por lo que permite que cualquier persona sin acceso a la herramienta de Matlab lo pueda utilizar.

Se utilizarán modulaciones y parámetros que se implementan en la actualidad en campos como la comunicación móvil, comunicación vía satélite, DVB-T, fibras móviles.

Como primer contacto con las modulaciones, el trabajo se centrará en modulaciones QAM y PSK llegando más allá con la implementación de OFDM.

La modulación PSK se utiliza en estándar de red LAN inalámbrica, entre otros:

- IEEE 802.11b en alguna de las velocidades de datos, con un número de subportadoras total de 64 y subportadoras de datos de 48 o de 128 y 108 respectivamente.
- Transmisores pasivos de bajo coste que se adapten a los diversos estándares de RFID como la norma ISO 14443 o Bluetooth 2.0.
- IEEE 802.15.4, estándar usado por ZigBee.

También se utiliza para transmisión de señales de alta definición en HDTV y en empresas satelitales para enviar señales de video en HD. En comunicaciones móviles se utiliza en EDGE (QPSK, 8-PSK), UMTS (QPSK y QAM) y LTE (BPSK, QPSK,16-QAM y 64-QAM junto a OFDM).

Las modulaciones más comunes son la BPSK y la QPSK, aunque la BPSK se ha sustituido mayoritariamente por la QPSK. Normalmente se suele obviar la modulación 8-PSK y pasar directamente a la 16-QAM, ya que la PSK con alto número de símbolos no funciona bien. Las ventajas que tiene frente a las modulaciones QAM es la simplicidad, ya que todos los símbolos tienen la misma potencia.

La modulación QAM se asocia a sistemas de transmisión de señales telefónicas, de televisión, satélite, modem ADSL. Las más comunes son 16-QAM y 64-QAM.

OFDM se usa en los siguientes sistemas DAB, DVB-T, ISDB-T, IEEE 802.11 a/g/n, Hiper LAN 2, televisión digital DVT-T, radio digital DAB, radio digital de baja frecuencia DRM, protocolo de enlace ADSL, IEEE 802.11 a/g, LAN, WiMax.

1.1. Estructura

El resto del presente documento está dividido en 5 capítulos:

- **Segundo capítulo**: se revisarán los principios de las comunicaciones digitales y los bloques para diseñar un sistema de radiocomunicación. Este capítulo es fundamental para poder realizar la simulación de un sistema de radiocomunicación.
- **Tercer capítulo**: se hace una introducción general a Python como toma de contacto con el lenguaje, ya que es necesario para implementar las simulaciones.
- **Cuarto capítulo**: se combinarán diseño e implementación para realizar las simulaciones de un sistema de radiocomunicación para las modulaciones lineales QAM y PSK. Se tendrán en cuenta los parámetros más comunes utilizados en sistemas de radiocomunicación. Además, se aplicará multiplexación por división ortogonal en frecuencia para mejorar la calidad del sistema.
- **Líneas futuras**: centradas sobre todo en implementaciones que se pueden realizar a partir de la realización del trabajo.
- **Conclusiones**: recapitulará todas las conclusiones que se hayan podido extraer a lo largo del trabajo y en la implementación del sistema.

1.2. Objetivos

El objetivo principal del trabajo es implementar un sistema de radiocomunicación utilizando como lenguaje de programación Python. De esta forma se profundizará en los conocimientos adquiridos en el grado a la vez que se coge soltura en un lenguaje de programación que está teniendo cada vez más importancia en el campo científico.

Para adquirir este objetivo habrá que realizar una implementación utilizando dos tipos de modulaciones: QAM y PSK. Se evaluarán dos casos de implementación, el primero consistirá en un transmisor con varias etapas: fuente de información, modulación y filtrado en transmisión, un canal AWGN con distintas relaciones E_b/N_o con las que evaluar la influencia del ruido en el sistema y por último un receptor que contiene un filtro en recepción acoplado al filtro en transmisión y un demodulador tras el que se medirá la tasa de error de bit (BER) del sistema. El segundo caso utilizará OFDM que implementará una variación en los bloques de transmisión y recepción y en el que se le da importancia a la implementación de la transformada discreta de Fourier, directa e inversa (FFT e IFFT).

Al final del trabajo se espera observar las diferencias entre la implementación de los distintos sistemas y contrastar los valores obtenidos por simulación de la BER con los valores teóricos. Además, se espera mostrar que se pueden conseguir mejoras de dichos valores en el caso de implementar el sistema con OFDM.

2 SISTEMAS Y SIMULACIÓN DE COMUNICACIONES DIGITALES

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

2.1. Variables y procesos aleatorios

2.1.1 Introducción

Para modelar un conjunto de señales aleatorias se usan variables y procesos aleatorios. En un determinado momento, es decir, en un momento instantáneo se puede modelar la respuesta mediante una variable aleatoria. Para un conjunto de señales que dependen del tiempo, la respuesta se modela mediante procesos aleatorios.

2.1.2 Variables aleatorias

Las variables aleatorias son funciones que tienen un valor asignado en un espacio. Se caracterizan por un conjunto de valores que describe una función distribución de probabilidad, $F_X(x)$, en el que x es el valor de la muestra para un caso en particular y X es la variable aleatoria.

$$F_X(x) = P(X \leq x) \tag{2–1}$$

Las variables aleatorias pueden ser continuas si su función de distribución es continua en todos los puntos o discreta si la función de distribución es constante excepto en discontinuidades de salto. Para caracterizar completamente una variable aleatoria se tiene que conocer la función de distribución y algunas medidas estadísticas como la media y la varianza. Definiendo su media y varianza como:

$$\mu_X = E\{X\} \tag{2–2}$$

$$\sigma_X = E\{(X - \mu_X)^2\} \tag{2–3}$$

Siendo $E\{g(X)\}$ el valor esperado calculado para variables continuas o discretas.

2.1.2.1 Variables continuas

La relación entre la función de distribución de probabilidad y la función de densidad es:

$$f_X(x) = \frac{dF_X(x)}{dx}, -\infty < x < \infty \quad (2\text{–}4)$$

x tiene infinitos valores en la recta real, por lo que habrá que integrar para todos los valores posibles de la recta real.

$$F_X(x) = \int_{-\infty}^{x} f_X(a)da \quad (2\text{–}5)$$

El valor esperado de una variable aleatoria continua se calcula como:

$$E\{g(X)\} = \int_{-\infty}^{\infty} g(x)f_X(x)dx \quad (2\text{–}6)$$

2.1.2.2 Variables discretas

La relación entre la función de distribución de probabilidad y la función de densidad es:

$$F_X(x) = P(X \leq x) = \sum_{\forall x_i \leq x} P(X = x_i) \quad (2\text{–}7)$$

Por lo que, para evaluar un suceso, hay que comprobar la probabilidad para cada caso concreto.

El valor esperado de una variable aleatoria discreta se calcula como:

$$E\{g(X)\} = \sum_i g(x_i)P(X = x_i) \quad (2\text{–}8)$$

2.1.3 Procesos aleatorios

Un proceso aleatorio es un conjunto de señales en el tiempo que se obtienen tras realizar un experimento, por lo que en un principio no se conoce la forma de onda que se obtendrá. En cada momento se asigna una función llamada función muestra que asigna cada resultado a una función real del tiempo. Los procesos se pueden caracterizar mediante sus funciones muestra o mediante el comportamiento estadístico de las variables aleatorias que lo componen. Los procesos aleatorios más importante son los estacionarios y los ergódicos.

2.1.3.1 Procesos aleatorios en el tiempo

Los procesos estacionarios y no estacionarios se usan para la simulación y el análisis de un sistema de comunicaciones. Si el proceso es estacionario entonces la respuesta y(t) ante una entrada x(t) desplazada un instante t_0 , x(t-t_0), tendrá la forma y(t-t_0)

Un proceso aleatorio X(t) se considera estrictamente estacionario si ante una variación en el dominio del tiempo no cambia sus propiedades. Para describir un proceso aleatorio estrictamente estacionario por su media y correlación estas no varían con una variacion en el tiempo original, lo que se conoce como estacionario en sentido amplio, WSS.

$$E\{X(t)\} = \mu_x \quad (2\text{–}9)$$

$$R_x(t_1, t_2) = E[X(t_1)X(t_2)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_1x_2f_{X(t_1),X(t_2)}\left(x_1,x_2\right)dx_1dx_2 \quad (2\text{–}10)$$

En la simulación y modelado de sistemas se usan modelos estacionarios la mayor parte del tiempo. De los procesos aleatorios se obtienen muestras y a partir de ellas se obtienen diversos valores como el nivel de potencia, las tasas de error etc. Si la función solo tienen una muestra, los parámetros del proceso se pueden obtener haciendo un promedio en el tiempo.

Los procesos aleatorios también pueden ser ergódicos si el promedio en el tiempo converge en los valores esperados del conjunto. En un proceso ergódico todas las señales tienen que ser aleatorias y

conociendo a una función del proceso, no se pueden conocer a las demás por lo que el promedio en el tiempo no va aportar ninguna información. Los procesos aleatorios Gaussianos que se usan para modelar señales y ruido en muchas aplicaciones se pueden comprobar que son ergodicos ya que un proceso aleatorio Gaussiano estacionario es ergódico que se cumple si $|\tau| < \infty$.

$$\int_{-\infty}^{\infty} |R_{XX}(\tau)| d\tau < \infty \quad (2\text{–}11)$$

2.1.3.2 Correlación y densidad spectral de potencia para un proceso estacionario alteario.

La descripción de señales deterministas en frecuencia se obtiene por la transformada de Fourier, sin embargo, no se pueden aplican a las formas de ondas aleatorias de cada miembro del conjunto de funciones porque puede no existir. Para un proceso aleatorio estacionario, la función Autocorrelación describe como de rápido cambia la señal en función del tiempo, si cae rápidamente a cero cambia rápidamente en función del tiempo. Mas allá de esto, si la función autocorrelación tiene componentes periódicas, el proceso tendrá componentes periódicas, por lo que se puede obtener información del contenido en frecuencia de un proceso aleatorio

Hay algunas propiedades que son muy importantes:

1. El valor cuadrático medio se da en el origen de la función de autocorrelación.

$$R_X(0) = E[X^2(t)] \quad (2\text{–}12)$$

2. La función de autocorrelación es una función par de τ, es decir, $R_X(\tau) = R_X(-\tau)$ (2–13)

3. La función de autocorrelación tiene su magnitud máxima en $\tau = 0$ por lo que

$$|R_X(\tau)| \leq R_X(0) \quad (2\text{–}14)$$

En el modelo de un sistema de comunicaciones digitales, el proceso aleatorio X(t) entrará por un filtro lineal invariante con el tiempo con una respuesta impulsiva h(t). En esta transmisión se realizará una convolución entre el proceso aleatorio de entrada y el filtro, quedando un nuevo proceso aleatorio como:

$$Y(t) = \int_{-\infty}^{\infty} h(\tau_1) X(t - \tau_1) d\tau_1 \quad (2\text{–}15)$$

donde la media de *Y*(t) teniendo en cuenta que la de *X(t)* es constante es:

$$\mu_Y(t) = \mu_X \int_{-\infty}^{\infty} h(\tau_1) d\tau_1 = \mu_X H(0) \quad (2\text{–}16)$$

Y la autocorrelación de Y(t) viene dada por

$$R_Y(\tau) = E\left[\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(\tau_1) h(\tau_2) R_X(\tau - \tau_1 + \tau_2) d\tau_1 d\tau_2\right] \quad (2\text{–}17)$$

Aplicando las propiedades de la autocorrelación, en $R_Y(0) = E[Y^2(t)]$

$$E[Y^2(t)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(\tau_1) h(\tau_2) R_X(\tau_2 - \tau_1) d\tau_1 d\tau_2 \quad (2\text{–}18)$$

Para medir la respuesta del fitro en el dominio de la frecuencia se utiliza la transformada de Fourier.

$$h(f) = \int_{-\infty}^{\infty} H(t) \exp(-j2\pi f \tau_1)\, d\tau_1 \quad (2\text{–}19)$$

La densidad espectral de potencia para el proceso aleatorio *X(t)* es:

$$S_X(f) = \int_{-\infty}^{\infty} R_X(\tau) \exp(-j2\pi f \tau)\, d\tau \quad (2\text{–}20)$$

Algunas propiedades interesantes son:

La densidad espectral a una frecuencia 0 es igual al área encerrada en la función autocorrelación del proceso aleatorio.

$$S_X(0) = \int_{-\infty}^{\infty} R_X(\tau)d\tau \quad (2\text{–}21)$$

El valor cuadrático medio es igual al área total bajo la gráfica de la densidad espectral de potencia.

$$E[Y^2(t)] = \int_{-\infty}^{\infty} S_X(f)df \quad (2\text{–}22)$$

La densidad espectral de potencia no puede ser negativa.

$$S_X(f) \geq 0 \quad (2\text{–}23)$$

Si el proceso aleatorio está compuesto por valores reales:

$$S_X(f) = S_X(-f) \quad (2\text{–}24)$$

La densidad espectral de potencia normalizada tiene propiedades que se pueden asociar con la función densidad de probabilidad

$$p_X(f) = \frac{S_X(f)}{\int_{-\infty}^{\infty} S_X(f)df} \quad (2\text{–}25)$$

2.1.4 Proceso gaussiano

La función densidad de probabilidad de un proceso gaussiano es la siguiente:

$$f_Y(y) = \frac{1}{(2\pi)^{1/2}\sigma_Y} exp\left[\frac{-(y-\mu_Y)^2}{2{\sigma_Y}^2}\right] \quad -\infty < y < \infty \quad (2\text{–}26)$$

Donde μ_Y indica la media y $\sigma_Y$la covarianza.

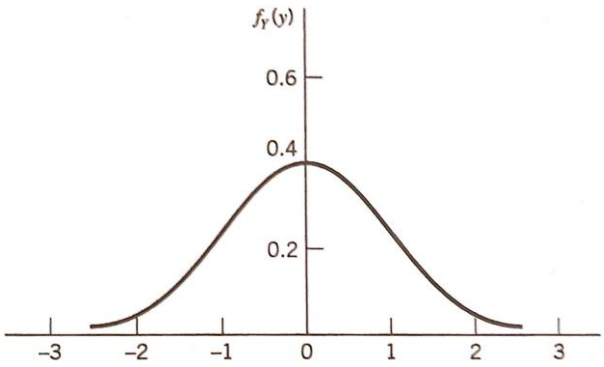


Figura 2-1. Distribución gaussiana normalizada

Las propiedades de los procesos alteatorios gaussianos son las siguientes:

- Si un proceso aleatorio X(t) es gaussiano, las variables aleatorias X(t_i) son conjuntamente gaussianas
- Si se conoce la media y la Autocorrelación, se tiene una descripción estadística completa del proceso.
- Si X(t_i) y X(t_j) son no correlacionadas para cualquier i y cualquier j, estas variables serán estadísticamente independientes.
- Si un proceso aleatorio gaussiano es WSS, es estacionario en sentido estricto.
- Si X(t) es un proceso aleatorio gaussiano e Y(t) es un proceso obtenido mediante una operación lineal de X(t), entonces Y(t) es gaussiano y conjuntamente gaussiano con X(t)

2.2. Sistemas de comunicación

El esquema básico de cualquier tipo de comunicación se compone por un **transmisor** del que sale la información que se quiere enviar por un **canal** hasta que llega a un destinatario que es el **receptor**. En comunicaiones digitales dicho esquema se puede representar de acuerdo al esquema que plantea Shannon:

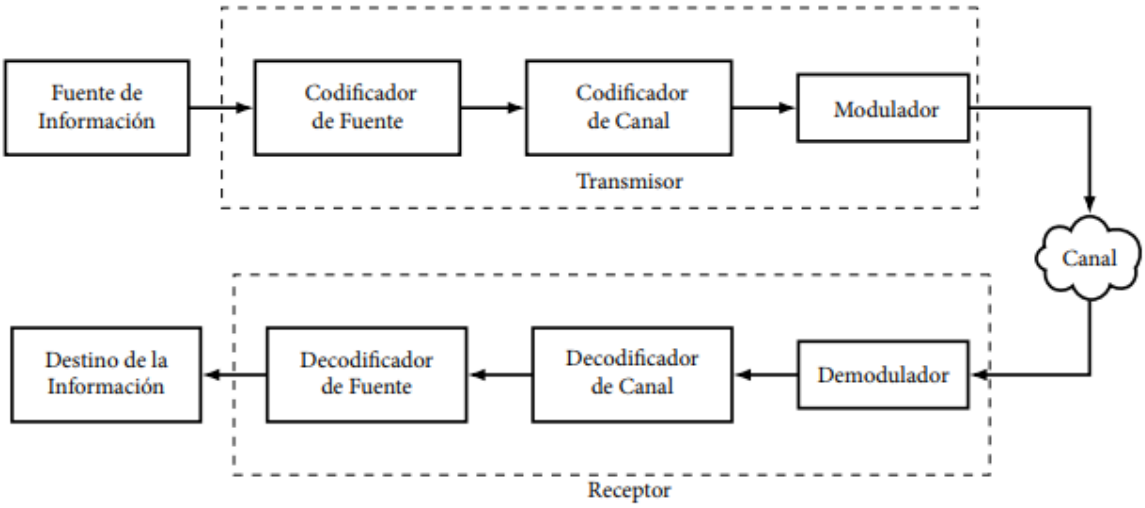


Figura 2-2. Sistema de comunicaciones digitales

2.2.1. Fuente de información y destino de la información

La fuente de información genera una señal que no se conoce a priori, modelada mediante muestras de un proceso aleatorio, a la que se conoce como mensaje que se desea conocer en el destino de la información. Al ser una señal no determinista no se sabe lo que se va a transmitir, por lo que siempre se transmite información.

2.2.2. Transmisor

2.2.2.1. Codificador de fuente

Su función principal es convertir una señal digital a una secuencia de dígitos binarios. Debe realizar una represetación eficiente de los símbolos generados por la fuente, lo que implica utilizar el mínimo número de bits que se pueda. Para ello, en este bloque se elimina la redundancia del mensaje.

La secuencia de bits a la salida de este bloque se conoce como *secuencia de información.*

2.2.2.2. Codificador de canal

Este bloque se encarga de transmitir la información de la forma más eficiente posible, es decir, con la menor probabilidad de error posible, lo que implica que haya el menor número de bits erróneos en la transmisión.

Hay distintos métodos para reducir la probabilidad de error como la codificación de bloques, codificación convolucional o la retransmisión de la información.

La tasa de transmisión varía ya que, aunque se haya quitado redundancia en el codificador de fuente, en este se añaden bits de redundancia para evitar errores.

2.2.2.3. Modulador

El modulador asocia la secuencia discreta de bits con un conjunto de señales en continua para su transmisión por el medio físico del sistema. Hay distintos tipos de modulaciones, por ejemplo, las binarias que consisten en transmitir 0's y 1's. Las señales generan un espacio de señal, por lo que para transmitir distintos símbolos estas deben ser diferentes. Generalizando, si se tiene k digitos binarios, se generan $M = 2^k$ señales distintas, s_i(t), con energía

$$E_i = \int_{-\infty}^{\infty} |s_i(t)| dt < \infty \tag{2–27}$$

2.2.3. Canal

El canal es un medio físico por el que se transmite la información que es inmutable, es decir, tiene unas características que no se pueden cambiar. Se puede someter a distintas fuentes que lo pueden alterar como los retrasos, atenuaciones o ruidos.

El ruído de la mayoría de los sistemas de telecomunicación se pueden describir atendiendo a lo que se conoce como **ruido aditivo blanco gaussiano (AWGN).**

El ruido es **aditivo** por lo que se suma a la señal que se está transmitiendo. Mientras sea un ruido bajo la señal recibida se parecerá a la original, sin embargo, si este es notable, puede llegar a distorsionar la señal tanto que no se parezca a la original y, por ende, impedir la comunicación.

Dicho ruido es un proceso aleatorio que se caracteriza de forma estadística, ya que no se puede predecir de forma exacta su comportamiento. Para estudiar el comportamiento del ruido se utiliza la distribución normal de Gauss. Una ventaja de esta distribución es que hay una probabilidad finita de que suceda un suceso por lo que, aunque la probabilidad sea muy pequeña, puede suceder.

El ruido **blanco** se extiende para todas las frecuencias, no obstante, en los sistemas de comunicaciones se limita en banda correspondiéndose a un ancho de banda igual al del canal de comunicación que lo deja pasar, atenuándose las frecuencias que quedan excluidas de dicha banda. En el dominio de la frecuencia su densidad espectral se ve como $S_w(f) = \frac{N_0}{2}$, donde $N_0 = KT_e$ siendo K la constante de Boltzmann y T_e la temperatura equivalente del receptor. En el dominio del tiempo, la densidad espectral constante da lugar a una autocorrelación con forma de delta, $R_w(t) = \frac{N_0}{2} d(\tau)$.

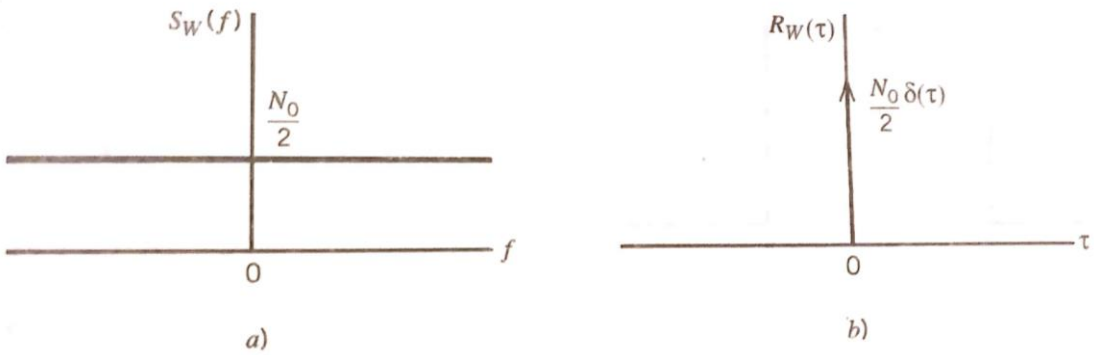


Figura 2-3. a) Representación de la densidad espectral del ruido. b) Representación de la autocorrelación del ruido.

El ruido blanco con media cero y densidad espectral $N_0/2$ limitado en banda por un filtro ideal paso de baja con un ancho de banda B queda como el mostrado en la Figura 2-4.

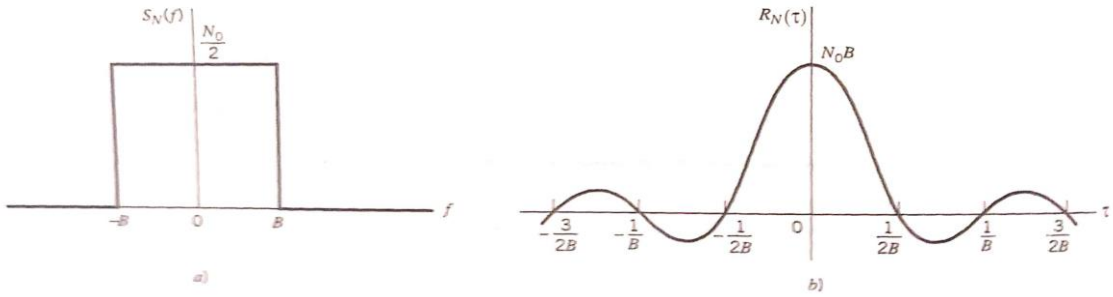


Figura 2-4. a) Representación de la densidad espectral del ruido limitado en banda. b) Representación de la correlación del ruido limitado en banda

Otros problemas del canal son la distorsión debida a la atenuación, causada por las características resistivas y y reactivas del canal físico, el retardo de grupo que se introduce debido a las distintas partes componentes de una señal que se propagan en el canal a velocidades diferentes y llegan al colector en un tiempo distinto de cero y por último, la distorsión por efectos meteorológicos que afecten al medio de propagación.

2.2.4. Receptor

2.2.4.1. Demodulador

En esta etapa se encuentran dos bloques: un receptor vectorial y un detector.

El receptor vectorial procesa la señal recibida, que está compuesta por la señal transmitida con variaciones debidas al canal. Este proceso implica transformar una señal paso de banda en una señal paso de baja. Además, pasa la señal de continua a discreta, obteniendo un vector de observación donde está recogida toda la información sobre la señal recibida para determinar el símbolo transmitido.

El detector se encarga de decidir que símbolo se ha transmitido en un tiempo determinado. Un símbolo tiene tantas posibilidades como señales se transmitan. Para ello, se tendrá que tener en cuenta el vector de observación, la probabilidad de transmisión de cada símbolo y una regla de decisión.

2.2.4.2. Decodificadores de canal y fuente

Hacen los procesos inversos del codificador de canal y fuente explicados anteriormente.

2.2.5. Filtrado en transmisión y recepción

En un sistema de comunicaciones digitales es necesario el filtrado para darle forma a la señal que se quiere transmitir, colocar la señal en las frecuencias determinadas o eliminar componentes indeseadas.

Los filtros de transmisión y de recepción se diseñan para minimizar la interferencia entre símbolos (ISI) en el sistema o para que sea nula, por lo que para la transmisión en banda base siguen el primer criterio de Nyquist. Al introducir la señal por un canal AWGN se introducirá ruido y distorsión por lo que la señal recibida y transmitida no serán iguales, por ello se tiene que utilizar un filtro de recepción acoplado a la señal recibida.

El primer criterio de Nyquist consiste en buscar un pulso p(t) que sea 0 para todos los múltiplos de nT excepto para n = 0, es decir

$$p(nT) = \begin{cases} 0 & n \neq 0 \\ 1 & n = 0 \end{cases} \qquad (2\text{–}28)$$

La entrada del detector se obtiene muestreando periódicamente cada T segundos el filtro acoplado de forma que la salida es:

$$y(mT) = p(0)a_m \sum_{n=-\infty}^{\infty} a_n p(mT - nT) + n(mT) \qquad (2\text{–}29)$$

a_m es el símbolo que se desea obtener en ese momento y n(mT) representa una variable aleatoria gaussiana.

En el dominio de la frecuencia el primer criterio de Nyquist puede definirse como:

$$\sum_{n=-\infty}^{\infty} P(w+nW) = T\,, con\ W = \frac{2\pi}{T} \quad (2\text{–}30)$$

Por lo que, por el primer criterio de Nyquist P(W) = 0 para $|w| > W_c$.

La frecuencia de Nyquist es: $W_0 = \frac{W}{2}$. (2–31)

El caso ideal será para el pulso de Nyquist donde $W_0 = W_c$ y el ancho de banda está ajustado para que, cuando termine un pulso, comience otro. Sin embargo, en la práctica esto es irrealizable, por lo que se tiene que dar la condición de $W_0 < W_c$, $T > \frac{\pi}{W_c}$ donde quedan los pulsos solapados y se pueden compensar los flancos de subida con los de bajada, algo que se conoce como teorema de banda lateral residual. Para ello, se utiliza la familia de los pulsos coseno alzado con la forma:

$$P(W) = \begin{cases} T & |w| \le W_0(1-\propto) \\ \frac{T}{2}\left[1+\cos\left(\frac{T}{2\propto}\left(|w| - \frac{\pi}{T}(1-\propto)\right)\right)\right] & W_0(1-\propto) \le |w| \le W_0(1+\propto) \\ 0 & |w| \ge W_0(1+\propto) \end{cases} \quad (2\text{–}32)$$

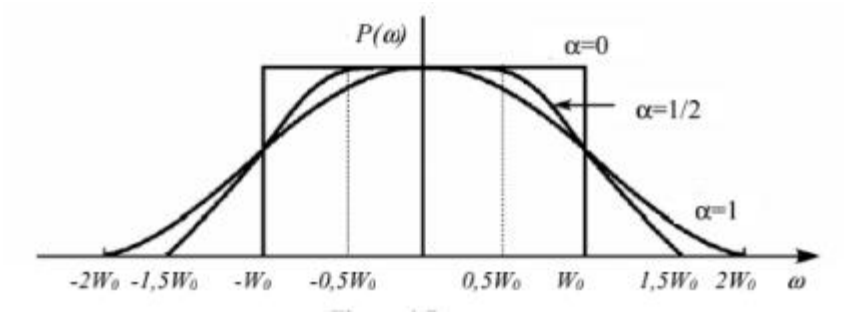


Figura 2-5. a) Representación del filtro coseno alzado para distintos valores de *roll-off*

Donde $\propto$ es el factor de *roll off* o factor de caída que indica la proporción de ancho de banda que se excede y toma valores entre 0 y 1, por lo que se necesitará un acho de banda de $W_c = W_0(1+\propto)$ como mínimo.

Hay tres formas de implementar filtros con la familia de coseno alzado, colocando el filtro coseno alzado en el transmisor o en el receptor y en el contrario un tren de deltas o repartiendo el coseno alzado entre transmisor y receptor, poniendo raíz de coseno alzado en ambos sitios. Los tres son equivalentes desde el punto de vista de la interferencia intersimbólica.

2.3. Simulación de procesos aleatorios

2.3.1. Estimación de la probabilidad de error de bit en un sistema digital

La probabilidad de error de bit (BER) proporciona la fiabilidad de un sistema digital. Por ejemplo, en un experimento hay un conjunto de sistemas iguales en los que las entradas y el ruido son distintos por lo que hay una forma de onda distinta para cada sistema, de modo que todas las formas de onda formarán un conjunto y la salida del sistema tras la decisión del símbolo tendrá que tener en cuenta algunos errores. Definiendo la tasa de error teniendo en cuenta que N es el número de símbolos transmitidos y n(N) es el número de errors cometidos, la tasa de error se tendría como $p = \lim_{N\to\infty} n(N)/N$. Tomando

un indicador de errores $e_k(t)$ en un tiempo t, $e_k(t) = 1$ si hay error y $e_k(t) = 0$ si no lo hay. De esta forma, si se compara un flujo de bits transmitido con el flujo de bits recibido, diviendo este error entre el número de bits totales transmitidos se obtiene la tasa de error de bit.

Para comprobar que los resultados prácticos se parecen a los teóricos, se dibujan dos tipos de curvas: la teórica que implementa la formula teórica de la BER en función de la E_b/N_o y el modelo semianalítico basado en la simulación cuasi-analítica con conocimientos a priori como la constelación que se está transmitiendo que permite demodular la señal dependiendo de la posición más cercana de forma analítica.

2.4. Modulación digital para un canal AWGN

Las modulaciones digitales son procesos en los que las señales digitales se transforman para que sean aptas al transmitirse por el canal. Se pueden clasificar en sistemas lineales o no lineales. Las primeras se modulan en la amplitud o en la fase portadora y en la segunda se modula en la frecuencia portadora.

La potencia y la eficiencia de una modulación depende del tamaño de la constelación formada por M símbolos, llamando a esta constelación M-aria. Cuanto mayor sea M, mayor será la cantidad de bits transmitidos en un ancho de banda, pero también será mayor la probabilidad de error producida por ruido y por desvanecimiento, lo que requerirá una SNR mayor si se desea tener una buena detección en el receptor.

El modelo general de señal digital es $x(t) = A(t)\cos(W_c \cdot t + \mathrm{W}(t) \cdot t + \phi(t))$, donde si se toma variable A(t) la modulación es en amplitud, si se toma variable W(t) es modulación en frecuencia y si es variable en $\phi(t)$ es modulación en fase.

Para modelar las señales dependiendo de su modulación, se utilizará un pulso conformador g(t) y una base del conjunto de señales unitarias ϕ_i , i=1, …, M.

En este caso, se tomarán las referencias de modulaciones para un canal de ruido aditivo y gaussiano, AWGN.

2.4.1. Modulación en amplitud (M-PAM)

La modulación por pulsos en amplitud consiste en transmitir cada uno de los símbolos por cada una de las M señales $s_i(t)$ y un pulso básico de energía unidad.

$$s_i(t) = s_i \cdot \phi(t) \tag{2–33}$$

Será banda base cuando el pulso $\phi(t)$ tenga la forma de la ecuación 2-34 donde $g(t)$ es un pulso conformador paso de baja de energía finita E_g.

$$\phi(t) = \frac{g(t)}{\sqrt{E_g}} \tag{2–34}$$

Será paso de banda, considerando la señal como ASK, cuando $\phi(t)$ tenga la forma de la ecuación 2-35

$$\phi(t) = \sqrt{\frac{2}{E_g}} \cdot g(t) \cdot \cos(w_c \cdot t) \tag{2–35}$$

El espacio de señal vendrá dado por la distancia mínima que hay entre los símbolos, de forma que

$$s_i = s_1 + d_{min}(i-1),\ i = 1,\ldots,M \tag{2–36}$$

La distancia mínima se calcula mediante la energía promedio de la constelación y un símbolo, ya que se mantiene entre puntos consecutivos de la constelación. Sabiendo la distancia al primer punto de la constelación

$$s_1 = -\frac{d_{min}}{2}(M-1) \tag{2–37}$$

Introduciéndolo en la ecuación 2-36 se obtiene que, de forma general:

$$s_i = \frac{d_{min}}{2}(2i - M - 1) \tag{2–38}$$

La energía promedio tiene la forma de:

$$E_{av} = \frac{d_{min}^{\ 2}}{12}(M^2 - 1) \tag{2–39}$$

Por lo que la distancia mínima entre dos símbolos consecutivos será:

$$d_{min} = \sqrt{\frac{12E_{av}}{M^2-1}} = \sqrt{\frac{12log_2 M}{M^2-1} E_{bav}} \tag{2–40}$$

Los coeficientes de la constelación se calcularán como:

$$s_i = \sqrt{\frac{3log_2 M}{M^2-1} E_{bav}}(2i - M - 1),\ i = 1\ldots M \tag{2–41}$$

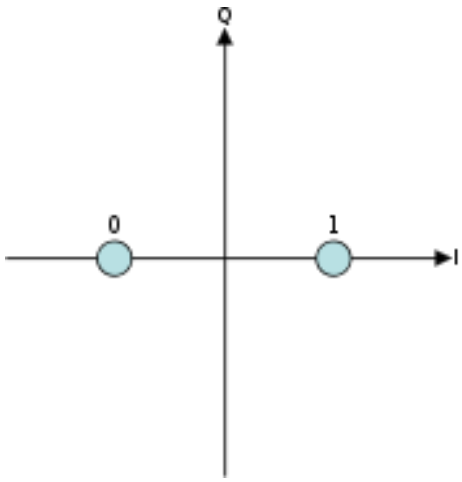


Figura 2-6. Representación de la constelación de una 2-PAM

2.4.2. Modulación en fase (M-PSK)

Las M señales paso de banda que se transmiten en este tipo de modulación tienen la forma de la ecuación 2-42 en la que $g(t)$ es un pulso conformador real paso de baja, de energía finita y duración como máximo de T.

$$s_i(t) = g(t)\cos\left(w_c \cdot t + \frac{2\pi(i-1)}{M}\right) \tag{2–42}$$

Mediante identidades trigonométricas y la ecuación 2-42, se pueden obtener las M señales en función del equivalente paso de baja de la señal transmitida tal que:

$$\begin{aligned} s_i(t) &= g(t)\cos\left(\frac{2\pi(i-1)}{M}\right)\cos(w_c \cdot t) - g(t)\,\mathrm{sen}\left(\frac{2\pi(i-1)}{M}\right)\mathrm{sen}(w_c \cdot t) \\ &= \mathrm{R}\left[g(t)\mathrm{e}^{\mathrm{j}\frac{2\pi(i-1)}{M}}\mathrm{e}^{\mathrm{j}w_c \cdot t}\right] = \mathrm{R}\left[g_{li}(t)\mathrm{e}^{\mathrm{j}\frac{2\pi(i-1)}{M}}\right] \end{aligned} \tag{2–43}$$

Esta modulación se considera lineal ya que las señales $s_i(t)$ varian linealmente dependiendo del valor de i que tomará valores de 1,…,M. Estará compuesta por una base ortonormal $[\phi_1, \phi_2]$ de dimensión 2 tal que:

$$\left\{\begin{array}{l} \phi_1 = \sqrt{\frac{1}{E}}g(t)\cos(w_c \cdot t) \\ \phi_2 = \sqrt{\frac{1}{E}}g(t)\sin(w_c \cdot t) \end{array}\right\} \tag{2–44}$$

Teniendo en cuenta las ecuaciones 2-43 y 2-44 se tiene que las señales transmitidas se puede expresar como:

$$s_i(t) = \sqrt{E}\cos\left(\frac{2\pi(i-1)}{M}\right)\phi_1(t) + \sqrt{E}\,\mathrm{sin}\left(\frac{2\pi(i-1)}{M}\right)\phi_2(t) \;\; con\, i = 1,\ldots,M \tag{2–45}$$

El espacio de señal tiene dos componentes que variarán en función del número de símbolos que se transmitan por lo que

$$s_i = [\sqrt{E}\cos\left(\frac{2\pi(i-1)}{M}\right), \sqrt{E}\sin\left(\frac{2\pi(i-1)}{M}\right)], \text{ con i} = 1,\ldots,\text{M}. \quad (2\text{–}46)$$

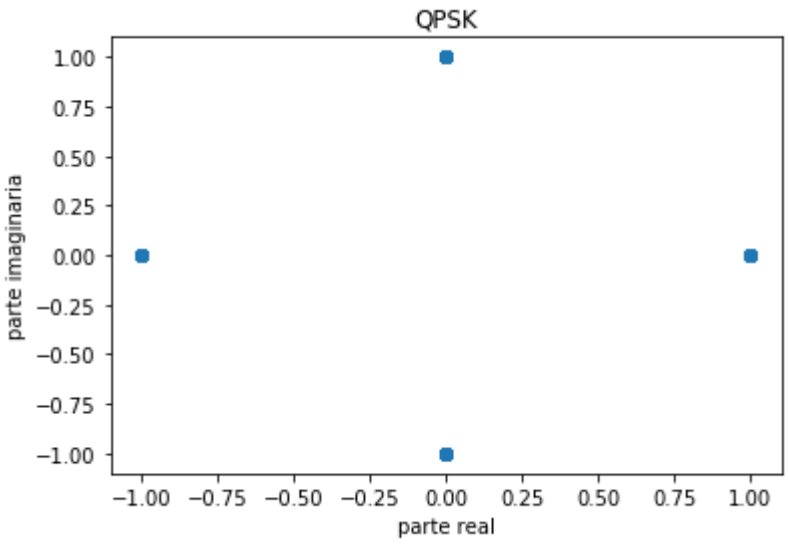


Figura 2-7. Representación de la constelación de una QPSK

2.4.3. Modulación en cuadratura (M-QAM)

La modulación en cuadratura combina las dos modulaciones vistas anteriormente, PAM y PSK. Las M señales paso de banda que se transmiten en este tipo de modulación tienen la forma de ecuación 2-47 en la que $g(t)$ es un pulso conformador paso de baja de la forma:

$$s_i(t) = \text{R}_i\text{g(t)}\cos(w_c \cdot t + \phi_i) \text{ con i} = 1,\ldots, \text{ M} \quad (2\text{–}47)$$

Donde el espacio de señal generado es:

$$\left\{ \begin{array}{ll} \phi_1 = \sqrt{\frac{2}{E_g}} g(t)\cos(w_c \cdot t) & 0 \le t \le T \\ \phi_2 = \sqrt{\frac{2}{E_g}} g(t)\sin(w_c \cdot t) & 0 \le t \le T \end{array} \right\} \quad (2\text{–}48)$$

Hay muchos tipos de constelaciones QAM, aunque las más utilizadas son las rectangulares que se pueden

considerar modulaciones PAM en cuadratura, de modo que con M_1-PAM o M_2-PAM se obtendría una $(M_1 x M_2)$-QAM. Por ejemplo, con dos 2-PAM se obtendría una 4-QAM. El espacio generado de señal queda como:

$$s_i(t) = a_{mI}\mathrm{g}(\mathrm{t})\cos(w_c \cdot t) - a_{nQ}\mathrm{g}(\mathrm{t})\mathrm{sen}(w_c \cdot t) \qquad (2\text{–}49)$$

Donde a_{mI} y a_{nQ} son las componentes de la constelación resultante.

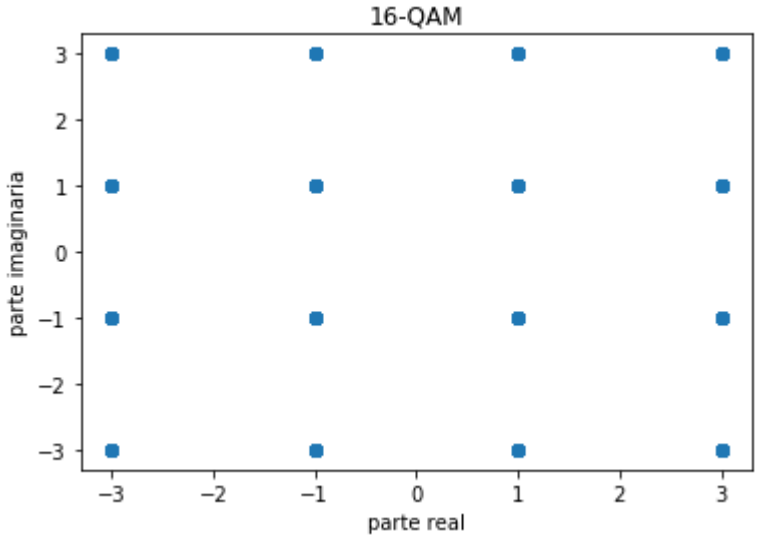


Figura 2-8. Representación de la constelación de una 16-QAM

2.4.4. Multiplexación por division ortogonal en frecuencia (OFDM)

OFDM permite repartir los datos entre N portadoras espaciadas entre sí a unas frecuencias determinadas, de forma que un canal se divide en N subcanales con una portadora por subcanal. A cada subcanal se le asocia un símbolo y se multiplexa en frecuencia. Para reducir costes se utiliza la transformada rápida de Fourier (FFT) y su inversa (IFFT) como técnica de procesamiento digital de señal.

OFDM se emplea porque permite utilizar de forma más eficiente el espectro ya que se divide el canal en subcanales de banda estrecha, es más resistente frente a desvanecimiento selectivo en frecuencia, elimina la ISI e ICI usando prefijo cíclico y al utilizar la FFT es muy eficiente computacionalmente en la modulación y demodulación.

Los parámetros que se han de tener en cuenta son:

- El número de subportadoras N, que depende del ancho de banda, la velocidad de los datos y la duración del tiempo útil. Será el número de puntos complejos que se procesan por la FFT.
- El esquema de modulación.
- El intervalo de guarda a utilizar.

Se tendrá en cuenta que se usa prefijo cíclico (CP), que la respuesta impulsiva del canal es menor que el CP, el canal es AWGN, hay sincronización entre el transmisor y el receptor y el desvanecimiento es bajo.

El esquema de una OFDM es más complejo que el esquema visto anteriormente de comunicaciones digitales en el que se incluye los siguientes bloques:

- Convertidor serie/paralelo: los símbolos en serie se pasan a un formato en paralelo.
- Convertidor paralelo/serie: los símbolos en paralelo se pasan a un formato en serie.
- Codificador: convierte los bits en símbolos dependiendo de la modulación M-QAM o M-PSK.
- Decodificador: convierte los símbolos recibidos en un flujo de bits.
- *Buffer*: almacena los N datos mapeados para poder introducirlos por la IFFT.
- IFFT: genera N portadoras a diferentes frecuencias ortogonales, generando la modulación OFDM.
- FFT: transforma una señal periódica en el dominio del tiempo en su equivalente espectro de frecuencia. Decoficándose con el algoritmo adecuado, se hace que los N subcanales vuelvan a ser uno.
- Conversor D/A: se produce una señal analógica en banda base que se modula en RF para transmitirse.

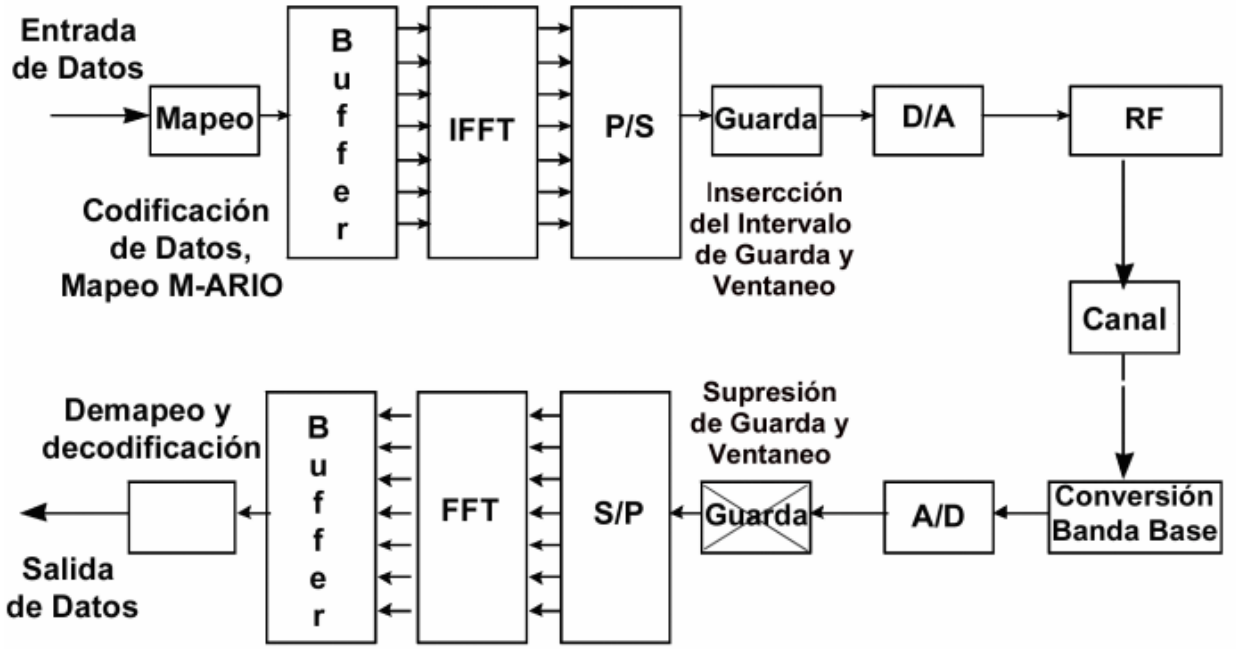


Figura 2-9. Esquema transmisión y recepción con modulación OFDM

La señal OFDM tiene la forma $s(t) = \sum_{m=-\infty}^{\infty}\left[\sum_{n=0}^{N-1} x_{n,m}\Phi_n(t-mT)\right]$, donde $\Phi_n(t)$ es un pulso rectangular modulado a la frecuencia portadora nW/N.

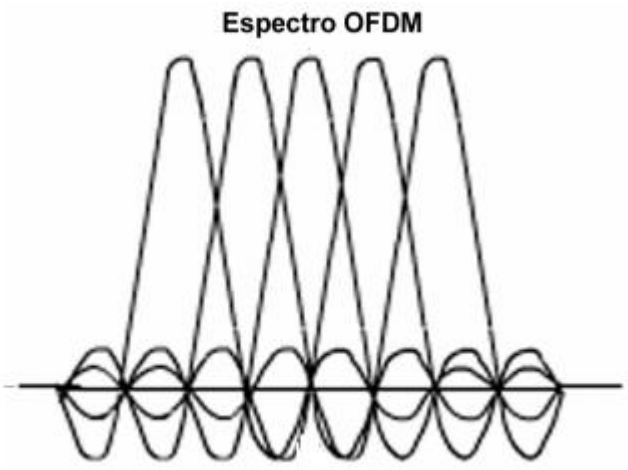


Figura 2-10. Espectro OFDM para N=5

2.5. Estimación teórica de la BER para un canal AWGN

2.5.1. Probabilidad de error en una señal modulada como M-PAM

Suponiendo que los símbolos son equiprobables, la probabilidad de error para una señal modulada como PAM será

$$P_M = \frac{1}{M}\sum_{i=1}^{M} P_e(s_i T_X) \tag{2–50}$$

Debido a que los símbolos están colocados en una recta real, se tendrán M-2 puntos que tengan dos puntos a su izquierda y derecha, uno que lo tenga solo a la derecha y otro que solo lo tenga a la izquierda. Considerando para un canal AWGN la probabilidad de error de un símbolo que solo tiene otro símbolo a la derecha o izquierda, por simetría ambas áreas van a ser iguales por lo que se puede modelar la probabilidad de error de dichos símbolos como:

$$P_e\left(s_i Tx\right) = \frac{1}{\sqrt{\pi N_0}} \int_{d_{min}/2}^{\infty} e^{-\frac{x^2}{N_0}} dx = Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right) \tag{2–51}$$

Para los símbolos internos que tienen dos símbolos a los lados

$$P_e(s_i T_X) = \sqrt{\frac{1}{\pi N_0}} \int_{d_{min}}^{\infty} e^{-\frac{x^2}{N_0}} dx = Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right) \tag{2–52}$$

La probabilidad de error media de símbolo se calcula teniendo en cuenta la probabilidad de error de cada símbolo por separado entre el número total de símbolos.

$$P_M = \frac{1}{M}\left\{(M-2)2Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right)+2Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right)\right\}=\frac{2(M-1)}{M}Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right) \quad (2\text{–}53)$$

La probabilidad de error equivalente (BER) en función de la relación energía de bit/ruido es:

$$P_b = \frac{P_M}{log_2 M}=\frac{2(M-1)}{Mlog_2 M}Q\left(\sqrt{\frac{E_{bav}}{N_0}\frac{6log_2 M}{M^2-1}}\right) \quad (2\text{–}54)$$

2.5.2. Probabilidad de error en una señal modulada como M-PSK

La probabilidad media de error es

$$P_M = P_e\left(s_i Tx\right) \quad (2\text{–}55)$$

La distancia mínima entre dos símbolos consecutivos de la constelación es:

$$d_{min} = 2\sqrt{E}\sin\left(\frac{\pi}{M}\right) \quad (2\text{–}56)$$

Como los símbolos tienen componente vertical y horizontal, el área que encierra la probabilidad de error se calculará con una integral doble.

$$P_M = \frac{1}{\pi N_0}\int_{-d_{min}/2}^{\infty}\int_{-\infty}^{\infty} e^{-\frac{x^2+y^2}{N_0}}dxdy = 2Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right) \quad (2\text{–}57)$$

Realizando la codificación de Gray, la probabilidad de bit equivalente es:

$$P_b = \frac{2}{log_2 M}Q\left(\sqrt{\frac{2log_2 ME_b}{N_0}}\sin\left(\frac{\pi}{M}\right)\right) \quad (2\text{–}58)$$

2.5.3. Probabilidad de error en una señal modulada como M-QAM

Para una modulación QAM rectangular se puede calcular la distancia mínima entre los símbolos como:

$$d_{min} = d = \sqrt{\frac{E_g}{2}(2A)^2} = A\sqrt{2E_g} \tag{2–59}$$

La energía promedio de la constelación viene dada por:

$$E_{av} = \frac{E_g}{2}\frac{A^2}{M}\sum_{m=1}^{M_1}\sum_{n=1}^{M_2}[(2m-1-M_1)^2 + (2n-1-M_2)^2] = \frac{E_g A^2 (M_1{}^2 + M_1{}^2 - 2)}{6} \tag{2–60}$$

Introduciendo esta energía promedio en función de la energía de pulso se tiene que la distancia es:

$$d_{min} = \sqrt{\frac{12E_{av}}{(M_1{}^2 + M_1{}^2 - 2)}} \tag{2–61}$$

Cada símbolo tendrá una componente en fase y otra en cuadratura, por lo que para demodular correctamente el símbolo tendrá que tenerse en cuenta la probabilidad media de error. Para ello se utiliza la cota de frontera más cercana:

$$N_e = 4\left(1 - \frac{M_1 + M_2}{2M}\right) \tag{2–62}$$

Teniendo en cuenta la probabilidad de error medio de una PAM, se tiene que:

$$P_M \leq N_e Q\left(\sqrt{\frac{d^2}{2N_0}}\right) = N_e Q\left(\sqrt{\frac{3log_2 M}{\frac{M_1{}^2 + M_2{}^2}{2M} - 1}\frac{E_{bav}}{N_0}}\right) \tag{2–63}$$

Para una modulación M-QAM cuadrada se tiene que M_1es igual que M_2 por lo que

$$P_M = 2Q\left(\sqrt{\frac{3log_2 M}{M-1}\frac{E_{bav}}{N_0}}\right) \tag{2–64}$$

3 INTRODUCCIÓN A PYTHON

We can only see a short distance ahead, but we can see plenty there that needs to be done

- Alan Turing-

Python es un lenguaje de programación libre creado en 1991 por Guido van Rossum. Las características más relevantes de este lenguaje son:

- Enfoque simple y efectivo a la programación orientada a objetos con una ejecución rápida.
- **Lenguaje interpretado de alto nivel**: los algoritmos se pueden entender de forma sencilla. El código es legible y reusable ya que tiene reglas de estilo para estandarizarlo.
- **Lenguaje multiplataforma**: se puede utilizar en diferentes sistemas operativos como Windows o Linux.
- **Lenguaje de tipado dinámico**: la corrección de la tipificación del código se hace mientras se está ejecutando en lugar de cuando se compila.
- **Lenguaje multiparadigma**: soporta otros lenguajes de programación como C/C++/Java.
- **Soporte de librerías**: por defecto, incorpora la librería estándar, pero se pueden añadir librerías para complementar a esta de forma que se introducen herramientas para desarrollo web, programación numérica, acceso de puertos serie, desarrollo de videojuegos y muchas aplicaciones más.
- **Integración de componentes**: los ficheros de Python pueden comunicarse fácilmente con otras partes de la aplicación mediante mecanismos de integración.

Este lenguaje es muy interesante por su extensa biblioteca estándar junto a todas las extensiones que se pueden añadir para completar el trabajo que se quiera realizar, ya sea desarrollo web, desarrollo de videojuegos, programación numérica y muchas aplicaciones más.

La clave de elegir Python entre otros lenguajes de programación como C/C++/Java es el tiempo que se ahorra evitando tener que realizar el proceso de compilación y comprobación de los errores.

3.1. Herramientas de desarrollo

3.1.1. Entorno de desarrollo

En este proyecto se utilizará el entorno de desarrollo integrado (IDE) Anaconda, Spyder. El objetivo principal de este IDE es proveer un entorno de programación en Python que se ejecuta de forma concurrente y asíncrona.

Incorpora librerías de Python útiles para incorporar métodos numéricos, gráficos o funciones. Además, ofrece tutoriales y referencias sobre Python y sobre Anaconda. Tiene a disposición entornos de programación como Spyder, Jupyter y otros.

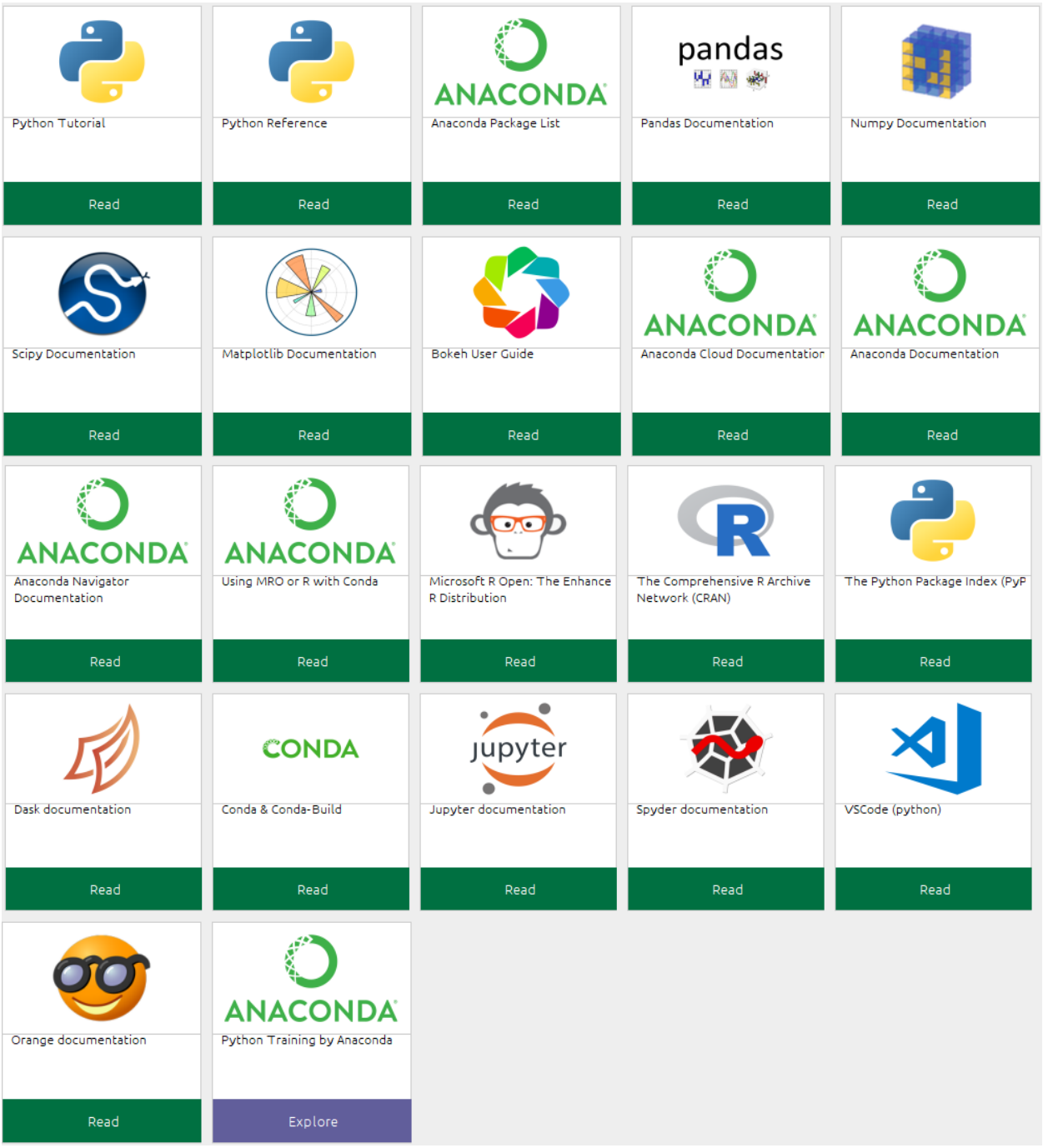


Figura 3-1. Documentación y referencias de Anaconda.

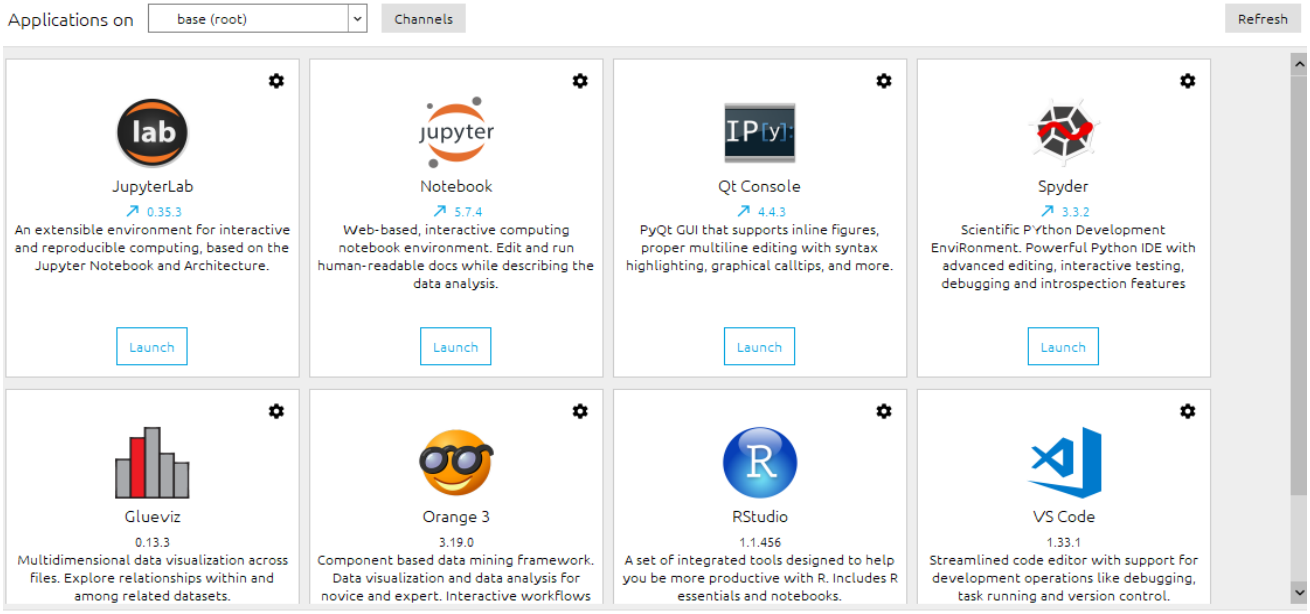


Figura 3-2Aplicaciones de entornos de desarrollo de Anaconda

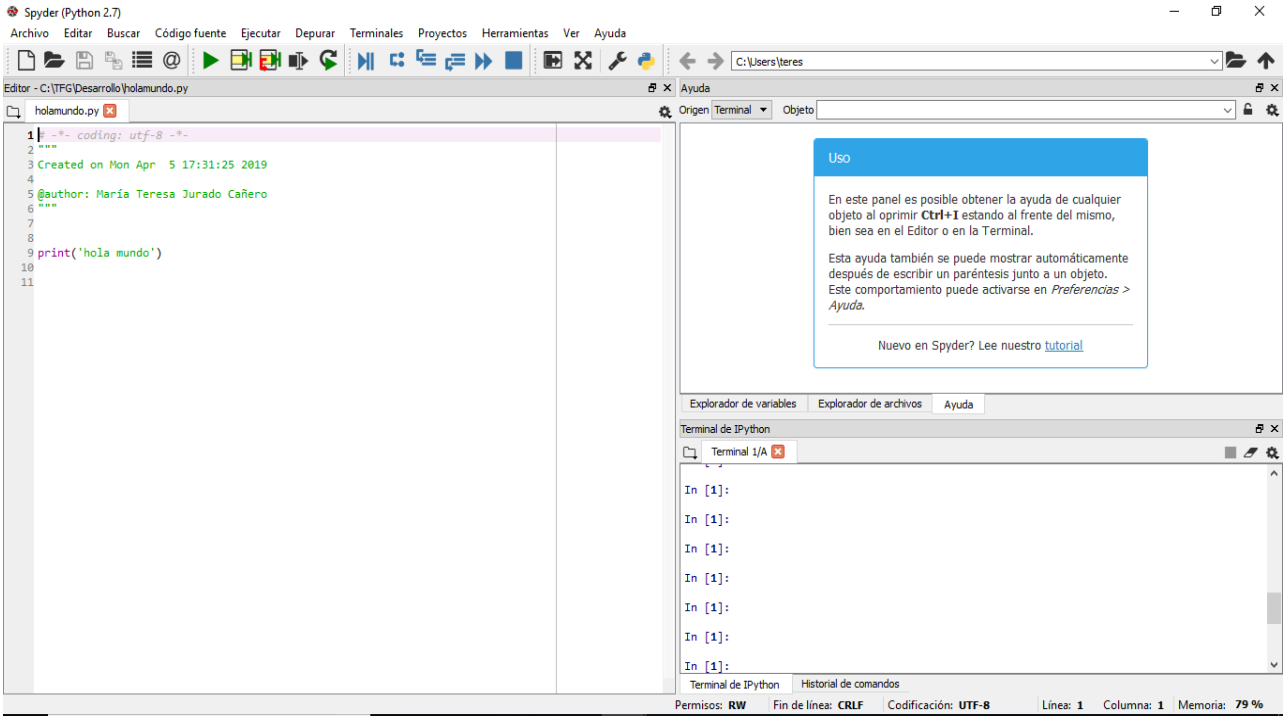


Figura 3-3Ventana de Spyder

En la ventana de Spyder aparecen los elementos rápidos que se pueden observar en la Tabla 3.1

Tabla 3–1. Elementos rápidos de Spyder

Función	Símbolo	Función	Símbolo
Nuevo archivo		Depurar archivo	
Abrir archivo		Ejecutar la línea seleccionada	
Guardar archivo		Ingresar en la función o método de la línea actual	
Guardar archivo como		Ejecutar hasta que la función o método actual termine	
Cambiar rápidamente entre archivos		Continuar con la ejecución hasta el siguiente punto de interrupción	
Búsqueda rápida de símbolos en el archivo		Detener la depuración	
Ejecutar archivo		Maximizar el panel actual	
Ejecutar la celda actual		Modo pantalla completa	
Ejecutar la celda actual y avanzar a la siguiente		Preferencias	
Ejecutar de nuevo el mismo archivo		Manejador de rutas de Python	

3.1.2. Modos de programación

Python es un lenguaje de programación que tiene dos modos con los que se puede trabajar:

3.1.2.1. Modo interactivo

Los comandos se escriben directamente en la terminal por lo que se irá respondiendo a los comandos individualmente conforme se van introduciendo.

Ejemplo 3–1. *Introducción de instrucciones directamente desde la pantalla*

```
In [1]: print('Imprimiendo un mensaje
directamente desde la consola')
Imprimiendo un mensaje directamente desde la
consola
```

3.1.2.2. Mediante scripts

Los comandos se escriben en un fichero terminado en .py y, una vez introducido todo el código perteneciente a dicho fichero, se compila y se comprueban los errores que se han obtenido en el caso de que haya o se obtienen respuestas conforme se ejecuta.

Ejemplo 3–2. *Introducción de instrucciones mediante un archivo*

```
In [1]: runfile('C:/TFG/Python/
impresion_desde_archivo.py', wdir='C:/TFG/Python')
Imprimiendo un mensaje directamente desde un script
```

El manejo de cualquier lenguaje de programación es aconsejable realizarlo utilizando archivos en la realización de cualquier proyecto, por lo que la primera forma de programación en Python es correcta pero no adecuada.

3.2. Conceptos básicos

3.2.1. Variables

Las variables son espacios donde se pueden guardar datos. Cada variable se caracteriza por ser de un tipo determinado y tener un valor asignado. Si se desea que una variable sea de un tipo se puede forzar a que esta lo sea.

3.2.1.1. Tipo int

Tipo al que se le pueden asignar números enteros.

Ejemplo 3–3. *Tipo int*

```
In [13]: entero=1; print(entero);print(type(entero));
1
<class 'int'>
```

3.2.1.2. Tipo float

Tipo al que se le pueden asignar números con decimales.

Ejemplo 3–4. *Tipo float*

```
In [14]: flotante=1.1; print(flotante);print(type(flotante));
1.1
<class 'float'>
```

3.2.1.3. Números complejos

Se definen como x=complex(a, b), siendo a la parte real y b la imaginaria. Para obtener los valores de la parte compleja e imaginaria se puede acceder como x.real, x.imag. Para obtener el módulo se usa abs(x).

Ejemplo 3–5. *Tipo complex*

```
complejo=complex(real=1,imag=1);
print('\n'+str(complejo));
print(type(complejo));
print('Parte real: '+ str(complejo.real));
print('Parte imaginaria: '+str(complejo.imag));
```

```
In [23]: runfile('C:/TFG/Desarrollo/complejos.py', wdir='C:/TFG/Desarrollo')

(1+1j)
<class 'complex'>
Parte real: 1.0
Parte imaginaria: 1.0

In [29]: abs(complejo)
Out[29]: 1.4142135623730951
```

3.2.1.4. Tipo bool

Variable a la que se le puede asignar el valor True o False.

Ejemplo 3–6. *Tipo bool*

```
In [28]: boleano=True;print(boleano);print(type(boleano))
True
<class 'bool'>


In [30]: boleano=False;print(boleano);print(type(boleano))
False
<class 'bool'>
```

3.2.1.5. Cadena de texto

Se pueden definir entre comillas simples o dobles, la ventaja de esta segunda es que facilita poder escribir apóstrofos entre los caracteres.

Ejemplo 3–7. *Tipo cadena de texto*

```
In [31]: cadena='Hola mundo';print(cadena);print(type(cadena));
Hola mundo
<class 'str'>
```

Las cadenas se pueden concatenar con el símbolo “+”. Si se desea que una subcadena se repita un número n de veces se debe poner n*subcadena en la concatenación. Para imprimir una cadena con distintos tipos de variables se tienen que poner de la siguiente forma:

Ejemplo 3–8. *Concatenación de cadenas de distinto tipo*

```
var1='uno'
var2= 1;
var3= 1.0;
print('El número '+var1+' es '+str(var2)+' como entero y es '+str(var3)+' como flotante')

In [10]: runfile('C:/TFG/Desarrollo/print_ejemplo.py', wdir='C:/TFG/Desarrollo')
El número uno es 1 como entero y es 1.0 como flotante
```

Las cadenas tienen una gran cantidad de métodos. Algunos de ellos son:

- str.capitalize() : devuelve una copia de la cadena original con la primera letra en mayúscula y el resto en minúscula.
- str.count(sub,start,end): devuelve el número de veces que se repite sub entre start y end.
- str.find(sub,start,end): devuelve el primer índice que contenga sub entre start y end.
- str.rfind(sub,start,end): devuelve el índice más alto que contenga sub entre start y end.
- str.index(sub,start,end): devuelve el índice de sub entre start y end. La diferencia con find es que da error cuando no encuentra ningún índice.
- str.join(iterable): devuelve la cadena str concatenada con el iterable.
- str.split(sep=None,maxsplit=1): devuelve una lista de palabras en la cadena, con sep como delimitador y maxsplit el máximo número de veces que se separa.
- str.isalnum(): devuelve True si la cadena es numérica completamente y False si algún carácter es distinto de un número.
- str.isalpha(): devuelve True si todos los caracteres de la cadena son caracteres del alfabeto y False en caso contrario.
- str.isascii(): devuelve True si la cadena está vacía o está compuesta de símbolos del código ascii y False en caso contrario.
- str.isdecimal(): devuelve True si la cadena es de caracteres decimales y False en caso contrario.
- str.isdigit(): devuelve True si en la cadena hay al menos un carácter y False en caso contrario.

3.2.2. Operadores aritméticos

Suma	**Negación**	**Exponente**	**División entera**
+	-	**	//
Resta	**Multiplicación**	**División**	**Módulo**
-	*	/	%

Ejemplo 3–9. *Operadores aritméticos*

```
#Ejemplo suma
suma=2+3;
print('La suma de 2+3 es: '+str(suma)+'\n');
#Ejemplo resta
resta=3-2;
print('La resta de 3-2 es: '+str(resta)+'\n');
#Ejemplo negación
Negacion=-1;
print('La negación de 1 es: '+str(Negacion)+'\n');
#Ejemplo multiplicación
Multiplicacion=2*2;
print('La multiplicación de 2*2 es: '+str(Multiplicacion)+'\n');
#Ejemplo exponente
Exponente=2**2;
print('2 elevado a 2 es: '+str(Exponente)+'\n');
#Ejemplo division
Division=3/2;
print('La división de 3/2 es: '+str(Division)+'\n');
#Ejemplo division_entera
Division_entera=3//2;
print('La división entera de 3/2 es: '+str(Division_entera)+'\n');
#Ejemplo modulo
Modulo=3%2;
print('El módulo de 3/2 es: '+str(Modulo)+'\n');

In [11]: runfile('C:/TFG/Desarrollo/Operadores_Aritméticos.py', wdir='C:/TFG/
Desarrollo')
La suma de 2+3 es: 5

La resta de 3-2 es: 1

La negación de 1 es: -1

La multiplicación de 2*2 es: 4

2 elevado a 2 es: 4

La división de 3/2 es: 1.5

La división entera de 3/2 es: 1

El módulo de 3/2 es: 1
```

3.2.3. Operadores lógicos

and	or	not
True/False	**True/False**	**True/False**
Todas las condiciones se cumplen	Al menos una condición se cumple	Negación

Ejemplo 3–10. *Operadores lógicos*

```
#Logica AND
condicion_and1=verdadero and falso
condicion_and2=verdadero and verdadero
condicion_and3=falso and falso
print('Con la logica and Verdadero y Falso es: '+str(condicion_and1)+ "\n"+
      'Con la logica and Verdadero y Verdadero es: '+str(condicion_and2)+"\n"+
      'Con la logica and Falso y Falso es: '+str(condicion_and3))

#Logica OR
condicion_or1=verdadero or falso
condicion_or2=verdadero or verdadero
condicion_or3=falso or falso
print('Con la logica or Verdadero y Falso es: '+str(condicion_or1)+"\n"+
      'Con la logica or Verdadero y Verdadero es: '+str(condicion_or2)+"\n"
      'Con la logica or Falso y Falso es: '+str(condicion_or3))

#Logica NOT
condicion_not1=not(verdadero)
condicion_not2=not(falso)
print('Con la logica not Verdadero es: '+str(condicion_not1)+"\n"+
      'Con la logica not Falso es:'+str(condicion_not2))
```

```
Con la logica and Verdadero y Falso es: False
Con la logica and Verdadero y Verdadero es: True
Con la logica and Falso y Falso es: False
Con la logica or Verdadero y Falso es: True
Con la logica or Verdadero y Verdadero es: True
Con la logica or Falso y Falso es: False
Con la logica not Verdadero es: False
Con la logica not Falso es:True
```

3.2.4. Operadores relacionales

Igual que	Menor que	Menor o igual que
==	<	<=
Distinto que	**Mayor que**	**Mayor o igual que**
!=	>	>=

3.2.5. Comentarios

- De una sola línea: #comentario
- De varías líneas: “””comentario”””

3.2.6. Tipos de datos complejos

3.2.6.1. Lista

Es una estructura de datos que pueden almacenar distintos tipos de datos. Se puede acceder a cada dato mediante indexado. Permite modificar los datos.

Ejemplo 3–11. *Listas*

```
In [20]: lista=['Comienzo',True,1];print(lista[0]);print(lista[1]);print(lista[2])
Comienzo
True
1

In [21]: lista[0]='Final'

In [22]: lista
Out[22]: ['Final', True, 1]
```

Las listas tienen los siguientes métodos:

- list.append(x): agrega un elemento x al final de la lista.
- list.extend(x): agrega todos los elementos de un array x.
- list.inster(i,x): agrega el elemento x en la posición i.
- list.remove(x): elimina un elemento que tenga el valor x.
- list.pop([i]): quita el elemento en la posición i y lo devuelve.
- list.clear: quita todos los elementos de una lista.
- list.index(x[start[,end]]): los elementos start y end son opcionales. El índice que devuelve depende del inicio y final establecido.
- list.count(x): devuelve el número de veces que esta x en la lista.
- list.sort(key=None,reverse=False): ordena los elementos de una lista.
- list.reverse(): invierte los elementos de una lista.
- list.copy: devuelve una copia de la lista.

3.2.6.2. Tupla

Es una estructura de datos que pueden almacenar distintos tipos de datos. Se puede acceder a cada dato mediante indexado. No permite modificar los datos.

Ejemplo 3–12. *Tuplas*

```
In [37]: tupla=('Comienzo',True,1);print(tupla[0]);print(tupla[1]);print(tupla[2])
Comienzo
True
1

In [38]: tupla[0]='Final'
Traceback (most recent call last):

  File "<ipython-input-38-a3c623339ba4>", line 1, in <module>
    tupla[0]='Final'

TypeError: 'tuple' object does not support item assignment
```

3.2.6.3. Diccionario

Es una variable que almacena datos que pueden ser de distintos tipos. Se puede acceder a cada dato mediante una clave para acceder a un valor.

Ejemplo 3–13. *Diccionarios*

```
In [28]: diccionario={'Comienzo':1,'Final':2}

In [29]: print(diccionario['Comienzo']);print(diccionario['Final'])
1
2
```

3.2.7. Estructuras de Control de Flujo

Una estructura de control es un bloque en el que se agrupa código y se va ejecutando línea por línea. Estas pueden ser condicionales o iterativas. En python es necesario una serie de espacios en blanco que permite dar estructura a los bloques de código, a lo que se le llama indexación de código. Otro aspecto de python con respecto a las estructuras es el encoding en que se indica la codificación de caracteres que se utilizará en los scripts.

Hay distintas estructuras de control:

- **Asignación múltiple**: en una sola instrucción se les puede asignar valores a varias variables
- **Estructuras de control de flujo condicionales:** evalúan mediante operadores lógicos y relacionales como se debe evaluar las condicionales.
- **Estructuras de control iterativas:** permiten evaluar iterativamente una condición en el mismo código mientras esta se cumpla.

3.2.7.1. Estructuras de control de flujo condicionales

if...else

Es una condición que permite evaluar un bloque de instrucciones dependiendo de las condiciones a las que se somete.

if condición:

Bloque que se ejecuta si la condición se cumple

else:

Bloque que se ejecuta si la condición no se cumple

Ejemplo 3–14. *Estructura de control de flujo if… else.*

```
In [52]: if (Verdadero==True):
    ...:     cadena=True;
    ...: else:
    ...:     cadena=False;
    ...:

In [53]: cadena
Out[53]: True
```

3.2.7.2. Estructuras de control iterativas

while

El bucle while permite evaluar un bloque de instrucciones mientras que unas condiciones específicas se cumplan.

while condición:

Bloque que se ejecuta mientras se cumpla la condición

Ejemplo 3–15. *Estructura de control iterativa while*

```
In [63]: ii=1;
    ...: while ii<10:
    ...:     ii=ii+1;
    ...:

In [64]: ii
Out[64]: 10
```

for

El bucle for permite evaluar un bloque de instrucciones cambiando el valor de la variable en cada iteración por el siguiente valor del elemento iterable empezando por el primero, teniendo en cuenta que el primer elemento de una lista, una tabla o un direccionaría se indexa con el “0”.

for variable **in** elemento iterable:

cuerpo del bucle

Ejemplo 3–16. *Estructura de control iterativa for*

```
In [69]: cadena=[1,2,3,4,5];

In [70]: for x in cadena:
    ...:     print(x)
    ...:
    ...:
1
2
3
4
5
```

3.2.8. Funciones

3.2.8.1. Definición de funciones

Las funciones en Python se definen con la instrucción *def*, un nombre descriptivo y el algoritmo que irá indexado por cuatro espacios. Una función no es ejecutada hasta que no se le invoca, algo que se hace simplemente por su nombre.

def nombre_descriptivo(parámetros):

    #4 espacios en blanco y el algoritmo

Los valores que acompañan a la función cuando se la invoca son distintos a los valores locales de la función, ya que estos últimos no se forpueden usar fuera de esta. Se puede distinguir entre las variables que pertenecen a la subrutina, **variables locales**, las variables que pertenecen al programa principal, **variables globales**, y las que pertenecen a un nivel superior que la subrutina pero no pertenecen al programa principal, **variables no locales.**

Ejemplo 3–17. *Definición de una función imprimiendo una variable global*

```
def suma_3_numeros(a,b,c): #Nombre descriptivo
    suma_local=a+b+c;#variable local
    return suma_local

#Variables globales
a=1;
b=2;
c=3;
#Variable de salida que te devuelve el return
suma_vglobal=suma_3_numeros(a,b,c)
print('La suma de '+ str(a)+'+'+ str(b)+'+'+str(c)+' es ' +str(suma_vglobal));

In [18]: runfile('C:/TFG/Desarrollo/suma_3_numeros.py', wdir='C:/TFG/Desarrollo')
La suma de 1+2+3 es 6
```

Ejemplo 3–18. *Definición de una función imprimiendo una variable local*

```
def suma_3_numeros(a,b,c): #Nombre descriptivo
    suma_local=a+b+c;#variable local
    return suma_local

#Variables globales
a=1;
b=2;
c=3;
#Variable de salida que te devuelve el return
suma_vglobal=suma_3_numeros(a,b,c)
#print('La suma de '+ str(a)+'+'+ str(b)+'+'+str(c)+' es ' +str(suma_vglobal));
print('La suma de '+ str(a)+'+'+ str(b)+'+'+str(c)+' es ' +str(suma_local));
```

```
  File "C:/TFG/Desarrollo/suma_3_numeros.py", line 20, in <module>
    print('La suma de '+ str(a)+'+'+ str(b)+'+'+str(c)+' es ' +str(suma_local));

NameError: name 'suma_local' is not defined
```

Los parámetros de las funciones pueden combinarse de varias maneras en las funciones:

- Una función podrá ser llamada con menos argumentos de los que espera cuando están definidos en la propia función, estos parámetros se llaman **parámetros por omisión** y no debe dejarse ningún espacio en blanco después de definirse.

  def función(param1,param2=‘parámetro 2 definido’):

Ejemplo 3–19. *Definición de una función con parámetros por omisión*

```
def suma_2_numeros(a,b=4): #Nombre descriptivo
    suma_local=a+b #Variable local
    return suma_local

#Variables globales
a=1
suma_vglobal=suma_2_numeros(a)
print('La suma de '+str(a)+' + un parametro por omisión es '+ str(suma_vglobal))
```

```
In [31]: runfile('C:/TFG/Desarrollo/suma_2_numeros_parm_definidos.py',
wdir='C:/TFG/Desarrollo')
La suma de 1 + un parametro por omisión es 5
```

- Los parámetros se pueden cambiar de orden dentro de la función si se pasan los valores como pares, es decir, se invoca la función de la siguiente manera: función(param1=‘parámetro 1’, param2=‘parámetro 2’), a estos parámetros se les conoce como **keywords.**

Ejemplo 3–20. *Definición de una función con parámetros como keywords*

```
def nombre_completo(nombre,apellido): #Nombre descriptivo
    persona=nombre+' '+apellido;
    return persona

persona=str(nombre_completo(apellido='Jurado',nombre='Teresa'))
print('Los datos introducidos pertenecen a '+persona);
```

```
In [32]: runfile('C:/TFG/Desarrollo/parametros_keyword.py', wdir='C:/TFG/Desarrollo')
Los datos introducidos pertenecen a Teresa Jurado
```

- Los **argumentos arbitrarios** son un número desconocido de argumentos que se pasan en forma de tupla. Se le pone un asterisco (*) antes de la tupla para indicar que se está apuntando a dicha variable de modo que se podrá acceder a cualquier elemento de la tupla. En la función, los primeros parámetros serán parámetros fijos y seguidos de estos irán los arbitrarios. Los valores arbitrarios también pueden ser pares clave=valor, de forma que si se quiere acceder a estos se deben usar dos asteriscos (**) antes de la variable arbitraria.

Ejemplo 3–21. *Definición de una función con argumentos arbitrarios en forma de tupla*

```
def nombre_completo(*args): #Nombre descriptivo

    nombre=args[0]+' '+args[1];
    print('Los datos introducidos pertenecen a '+ str(nombre));

    return

nombre_completo('Teresa','Jurado')
```

```
In [40]: runfile('C:/TFG/Desarrollo/param_tuplas.py', wdir='C:/TFG/Desarrollo')
Los datos introducidos pertenecen a Teresa Jurado
```

Ejemplo 3–22. *Definición de una función con argumentos arbitrarios en forma de diccionario*

```
def nombre_completo(**kwargs): #Nombre descriptivo

    print('Los datos introducidos pertenecen a '+str(kwargs['nombre'])+' '+str(kwargs['apellido']));

    return

nombre_completo(nombre='Teresa',apellido='Jurado')
```

```
In [39]: runfile('C:/TFG/Desarrollo/param_dic.py', wdir='C:/TFG/Desarrollo')
Los datos introducidos pertenecen a Teresa Jurado
```

3.2.8.2. Funciones recursivas

Las funciones recursivas se contienen a sí mismas en el bloque de instrucciones que conforma su función. Pueden ser útiles en algunos casos, sin embargo, pueden caer en iteraciones infinitas y provocar errores en el programa. Por esta razón, deberán utilizarse solo cuando sea estrictamente necesario.

Ejemplo 3–23. *Función recursiva*

```
def recursiva(param):
    print('Param vale= '+str(param));
    if(param<3):
        print('Como el parámetro es menor que 3, se llama a la función recursiva')
        param=param+1;
        recursiva(param);
    else :
        print('La función recursiva ha terminado con parámetro= '+str(param));
        return


recursiva(1)

In [41]: runfile('C:/TFG/Desarrollo/recursiva.py', wdir='C:/TFG/Desarrollo')
Param vale= 1
Como el parámetro es menor que 3, se llama a la función recursiva
Param vale= 2
Como el parámetro es menor que 3, se llama a la función recursiva
Param vale= 3
La función recursiva ha terminado con parámetro= 3
```

3.2.8.3. Funciones destacadas

- round: redondea al entero más próximo. Si recibe dos argumentos redondea al decimal más cercano al segundo argumento.
- floor: redondea al entero inferior.
- abs: valor absoluto de un número.
- max: valor máximo de un array.
- min: valor mínimo de un array.
- sum: suma de todos los números de un array.
- sorted: ordena de menor a mayor el conjunto de valores de un array.
- arange(start,stop,step): devuelve un array de start hasta stop con pasos intermedios step.
- print: imprime por pantalla distintos tipos de variables.

3.2.9. Módulos, paquetes y namespaces

En Python, a los archivos se le coloca la extensión .py de forma que se pueden incorporar en paquetes. Para que un paquete sea considerado como tal, debe tener un archivo de inicio que se llame __init__.py que debe estar vacío.

Si se va a utilizar un paquete es necesario importarlo donde se quiera utilizar, mediante el comando **import** *archivo* sin colocar la extensión .py.

Se puede acceder a un elemento de un módulo importado siguiendo la ruta o namespace que se consigue añadiendo contenido por puntos, es decir **import** *paquete.elemento*. También se puede importar un elemento utilizando la forma **from** *paquete* **import** *elemento*. No se recomienda importar todos los elementos de un paquete, pero si se desea, se puede hacer de la forma **from** paquete **import** *

Para reducir el namespace se utilizan los alias de la forma **import** *elemento* **as** *e*.

3.2.10. Entrada/Salida de datos

3.2.10.1. Entrada por teclado

El método input permite obtener datos del exterior desde el teclado.

3.2.10.2. Salida de datos

El método output permite mostrar datos por pantalla.

3.2.10.3. Ficheros: lectura y escritura de datos.

Desde un fichero se pueden leer o escribir datos con las palabras reservadas with y as junto al método open.

```
with open("fichero",mode= "modo", encoding= "codificación") as
fichero:
bloque de instrucciones
```

En fichero se escribe el nombre del fichero junto con su extensión correspondiente, por ejemplo, datos.txt. El modo puede ser de lectura y escritura pero va más allá de esto, por lo que los distintos modos de apertura del fichero se verán reflejados en la Tabla 2-2 junto con la ubicación del puntero.

Tabla 3–2. Modos de apertura de un fichero

Indicador	Modo de apertura	Ubicación del puntero
r	Solo lectura	Inicio del archivo
rb	Solo lectura en modo binario	Inicio del archivo
r+	Lectura y escritura	Inicio del archivo
rb+	Lectura y escritura en modo binario	Inicio del archivo
w	Solo escritura. Si hay algo escrito en el fichero lo sobrescribe, si no, lo crea.	Inicio del archivo
wb	Solo escritura en modo binario. Si hay algo escrito en el fichero lo sobrescribe, si no, lo crea.	Inicio del archivo
w+	Lectura y escritura. Si hay algo escrito en el fichero lo sobrescribe, si no, lo crea.	Inicio del archivo
wb+	Escritura y lectura en modo binario. Si hay algo escrito en el fichero lo sobrescribe, si no, lo crea.	Inicio del archivo
a	Agrega contenido. Si no hay ningún archivo creado, lo crea.	Si el archivo existe se coloca al final. Si no existe al principio
ab	Agrega contenido en modo binario. Si no hay ningún archivo creado, lo crea.	Si el archivo existe se coloca al final. Si no existe al principio

a+	Agrega y lee contenido en modo binario. Si no hay ningún archivo creado, lo crea.	Si el archivo existe se coloca al final. Si no existe al principio
ab+	Agrega y lee contenido en modo binario. Si no hay ningún archivo creado, lo crea.	Si el archivo existe se coloca al final. Si no existe al principio

3.2.11. Programación Orientada a Objetos (POO)

3.2.11.1. Elementos principales

Clases

Una clase es la plantilla general de un objeto que incorpora métodos y variables iniciales de estado. En Python se utiliza la palabra *class* para definir una clase seguido del nombre de la clase.

Objetos

Los objetos son la forma de materializar una clase, es decir, rellenar los campos y variables de la plantilla inicial. Los objetos están compuestos por **propiedades** en las que están las características de dicho objeto y los **métodos** con los que se realizan operaciones con el objeto.

3.2.11.2. Herencia

La programación orientada a objetos tiene una estructura jerárquica en la que los nodos principales, llamados nodos padres, se subdividen en otros nodos, llamados nodos hijos. Los nodos hijos heredan todas las propiedades y métodos de los nodos padres.

3.3. Simulación de sistemas de comunicaciones digitales en Python

3.3.1. Paquetes específicos para la simulación de sistemas de comunicaciones

Para simular sistemas de comunicaciones se usa la librería CommPy que es libre y sirve para implementar algoritmos de comunicaciones digitales en Python.

Esta librería puede llevar a cabo diversas funciones como:

- Codificación de canal
- Modelado del canal
- Filtrado
- Deterioro de la señal
- Modulación y demodulación
- Secuencias

3.3.1.1. Codificación de canal

Para implementar la codificación de canal hay que añadir el paquete commpy.channelcoding.

Algunas de sus aplicaciones son:

- Códigos convolucionales del codificador que proporcionan porcentajes y matrices perforadas (*puncture matrices*).

  import commpy.channelcoding.conv_encode

  import commpy.channelcoding.Trellis

- Decodificador Viterbi para códigos convolucionales de *Hard Decision*.

  import commpy.channelcoding.viterbi_decode

- Decodificadores de mapas para códigos convolucionales basados en el algoritmo BCJR.

  import commpy.channelcoding.map_decode

- Codificador para un porcentaje de 1/3 sistematicamente concatenado en paralelo con Turbo Código.

  import commpy.channelcoding.turbo_encode

- Campos de Galois binarios (GF(2^m)) con polinomios y clases laterales ciclotómicas.

  import commpy.channelcoding.GF

- Creación de cualquier polinomio generador para un código cíclico de (n,k).

- Intervalos aleatorios de desentrelazadores.

  import commpy.channelcoding.RandIntervl

3.3.1.2. Modelado del canal

Para implementar el modelado de canal hay que añadir el paquete commpy.channels.

Algunas de sus aplicaciones son:

- Canal binario simétrico (BSC)

  import *commpy.channels.bsc*

- Canal binario (BEC)

  import *commpy.channels.bec*

- Canal binario AWGN (BAWGNC)

  import *commpy.channels.awgn*

3.3.1.3. Filtro conformador de pulsos

Para implementar el filtro conformador de pulsos hay que añadir el paquete commpy.filters.

Algunas de sus aplicaciones son:

- Filtro rectangular

  import *commpy.filters.rectfilter*

- Filtro Coseno alzado

  import *commpy.filters.rcosfilter*

- Filtro Raiz de Coseno alzado

  import *commpy.filters.rrcosfilter*

- Filtro Gaussiano

  import *commpy.filters.gaussianfilter*

3.3.1.4. Modulación/Demodulación

- Modulación de Fase (PSK)

  import *commpy.modulation.PSKModem*
- Modulación de amplitud en cuadratura (QAM)

  import *commpy.modulation.QAMModem*
- Procesado de la señal en recepción o transmisión utilizando MIMO

  import *commpy.modulation.mimo_ml*
- Procesado de la señal en recepción o transmisión OFDM

  import *commpy.modulation.ofdm_tx*

  import *commpy.modulation.ofdm_rx*

3.3.1.5. Otras funciones de interés

- Pasar de decimal a un array de bits o viceversa

  import *commpy.utilities.dec2bitarray*

  import *commpy.utilities.bitarray2dec*
- Distancia de Hamming

  import *commpy.utilities.hamming_dist*
- Distancia Euclidea

  import *commpy.utilities.euclid_dist*
- Tomar muestras adicionales

  import commpy.utilities.upsample

3.3.2. Otros paquetes de Python utilizados para la simulación de sistemas de comunicaciones

El paquete Commpy tiene todas las herramientas necesarias para evaluar sistemas de comunicaciones digitales pero hace falta utilizar otras librerías como NumPy, SciPy and Matplotlib.

3.3.2.1. NumPy

NumPy es una librería para cálculos científicos en Python que permite utilizar vectores N-dimensionales, funciones sofisticadas, algebra lineal, transformadas de Fourier y funciones con números aleatorios. Será necesario incorporar números aleatorios, números reales, imaginarios e intervalos específicos, la transformada rápida de Fourier (FFT) y su inversa (IFFT).

from *numpy.random* **import** *randint*

Ejemplo 3–24. *Ejemplo array aleatorio de ceros y unos*

```
In [49]: from numpy.random import randint

In [50]: randint(0,2,10)
Out[50]: array([1, 1, 1, 0, 1, 1, 1, 0, 0, 1])
```

from *numpy* **import** *real,imag,arrange*

Ejemplo 3–25. *Complejos*

```
In [53]: from numpy import real,imag

In [54]: a=complex(2,3)

In [55]: a.real
Out[55]: 2.0

In [56]: a.imag
Out[56]: 3.0
```

from *numpy.fft* **import** *fft, ifft*

3.3.2.2. SciPy

SciPy es una librería que proporciona rutinas numéricas eficientes de integración y optimización. En este caso, será de utilidad para cálculo de probabilidades y de funciones frecuentes en las comunicaciones digitales. Sería adecuado incorporar toda esta librería. Algunos métodos que son interesantes de destacar son linspace(start,stop,n_points) que crea un array de start a stop con n_points o por ejemplo las funciones trigonométricas seno y coseno. La función erfc, función de error de gauss, sirve para medir el error cometido en una función.

from *scipy* **import** *

from *scipy.especial* **import** *erfc*

3.3.2.3. Matplotlib

Matplotlib es una librería que incorpora herramientas para dibujar gráficas, de modo que para la representación de las gráficas de la BER frente a la Eb/No, se usará el objeto pyplot.

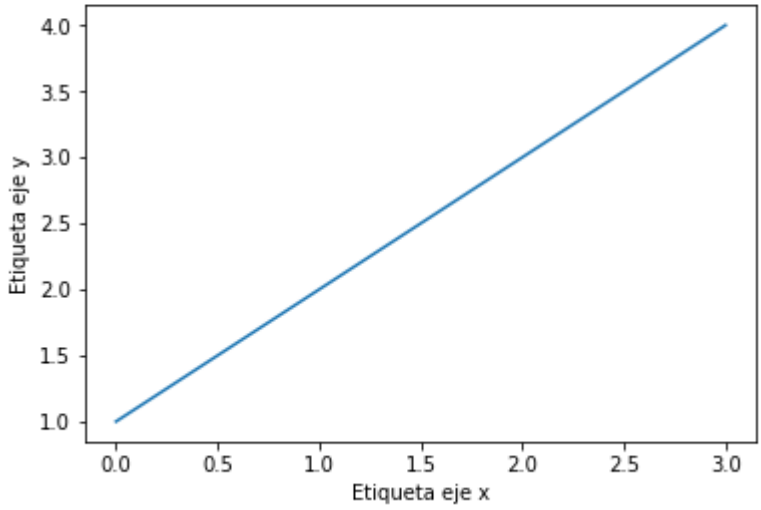
import *matplotlib.pyplot* **as** *plt*

Representación básica de gráficas

Para la representación de gráficas en dos dimensiones se utiliza el método plot(x,y,fmt), donde x e y son los vectores correspondientes al eje horizontal y vertical, fmt es una cadena opcional que nos permite establecer la forma de representación, por ejemplo “o” daría como resultados puntos en las coordenadas pertenecientes a x e, por defecto es una línea continua. El método plot tiene otras propiedades como son linewidth o ancho de línea, linestyle o estilo de gráfica (color y tipo) y marker que es un marcador que se pone en los puntos.

Ejemplo 3–26. *Representacion básica de gráficas*

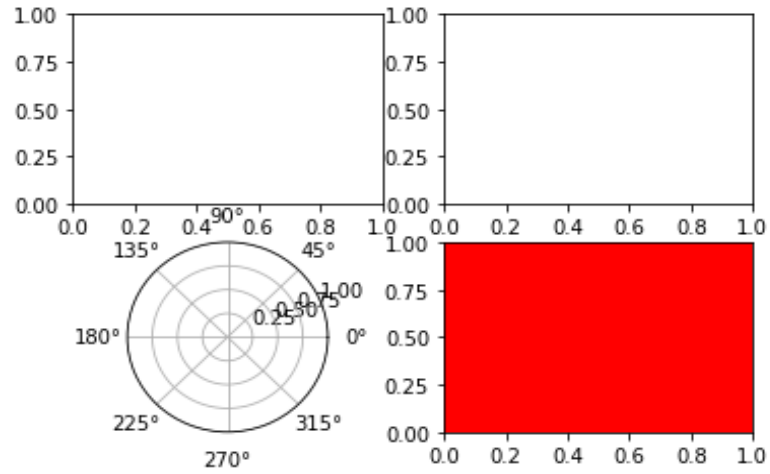
```
In [7]: import matplotlib.pyplot as plt
   ...: plt.plot([1,2,3,4])
   ...: plt.ylabel('Etiqueta eje y')
   ...: plt.xlabel('Etiqueta eje x')
   ...: plt.show()
   ...:
   ...:
```



Si se desea representar más de una gráfica en la misma ventana, se debe usar el método subplot(numRows,numCols,plotNum) en vez del plot convencional. En este método numRows es el número de filas, numCols es el número de columnas y plotNum el número de gráficas dentro de una misma figura.

Ejemplo 3–27. *Distintos tipos de gráficas representadas con subplot*

```
In [43]: ax1=plt.subplot(2,2,1);ax2=plt.subplot(2,2,2);
ax3=plt.subplot(223,projection='polar');ax4=plt.subplot(224,
sharex=ax1, facecolor='red')
```



Crear figuras

En la representación de gráficas, se pueden establecer nuevas figuras o ventanas para la representación de estas y que no se sobrescriban a las anteriores, ya que si no se utilizase esta opción, siempre estarían en la figura 1. Esta orden tiene la forma figure(num,figsize,dpi,facecolor,edgecolor,frameon), donde num es el número de figura, figsize el ancho y alto de la figura en pulgadas, dpi es la resolución de la imagen en pulgadas, facecolor es el color del rectángulo de la figura, edgecolor es el color del perímetro de la figura y frameon es un boleano que si esta activado es True y si no, se elimina el marco de la figura.

Otros métodos de interés

- show: presenta figuras en pantalla.
- legends: introduce una leyenda en las gráficas.
- xlabel: etiqueta del eje de coordenadas de abscisas de la gráfica actual.
- ylabel: etiqueta del eje de coordenadas de ordenadas de la gráfica actual.
- close: cierra la gráfica.

4 SIMULACIÓN DE UN SISTEMA DE RADIOCOMUNICACIÓN EN PYTHON

Este capítulo se dedicará al estudio de la BER frente a la relación E_b/N_o dependiendo de distintas condiciones en el sistema de radiocomuniación. Primero, se comprobarán qué tipo de modulaciones son las más adecuadas para que la BER sea la mínima posible puesto que se considerarán condiciones objetivo cuando esta tenga un valor de 10^{-6}. Dentro de los distintos tipos de modulaciones se utilizará un filtro de raíz de coseno alzado con distintos factores de *roll off*. Tras estudiar detenidamente todos los casos, se llegará a distintas conclusiones como cuáles son las modulaciones que minimizan la tasa de error de bit, qué factores de *roll off* son más adecuados en cada caso y las ventajas o desventajas de utilizar un filtro u otro.

El cálculo de la BER por simulación se realiza de igual forma, tanto para la modulación PSK como para la modulación QAM. Sin embargo, la BER teórica toma expresiones distintas. Ambas se miden una vez demodulada la señal en el receptor.

El código principal sigue la estructura de un sistema de comunicaciones digitales completo desde los bits que se desean transmitir hasta los bits que se reciben.

1. Creación de un intervalo de evaluación con las relacionas de energía de bit entre el ruido que se deseen evaluar.
2. Creación de un objeto para gestionar las modulaciones PSK o QAM.
3. Crear variables vacías para guardar los distintos valores de la BER teórica y la BER práctica.
4. Tener en cuenta todos los parámetros que se vayan a utilizar en el filtro. Los valores más comunes de *roll off* son 0.22 y 0.35, pero se añadirán también 0.5 y 0.8 para estudiar los distintos casos. Una vez especificados los parámetros, diseñar el filtro.
5. Crear una ristra de bits aleatoria para el estudio.
6. Modular dicha ristra con el objeto de la modulación correspondiente.
7. Sobremuestrear y filtrar los datos de entrada.
8. Pasar la señal filtrada por un canal AWGN.
9. Una vez haya pasado por el canal, se filtra y se submuestrea.
10. Para terminar en el receptor, se demodula la señal y se comprueban los errores que hay de transmisión.

4.1 Estudio previo teórico general para modulaciones M-arias

Primeramente, se va a realizar un estudio previo teórico con el fin de poder entender mejor las simulaciones realizadas en posteriores apartados. La gráfica 4-1 representa las curvas teóricas de la probabilidad de error de dos modulaciones M-arias: la modulación QAM y la modulación PSK.

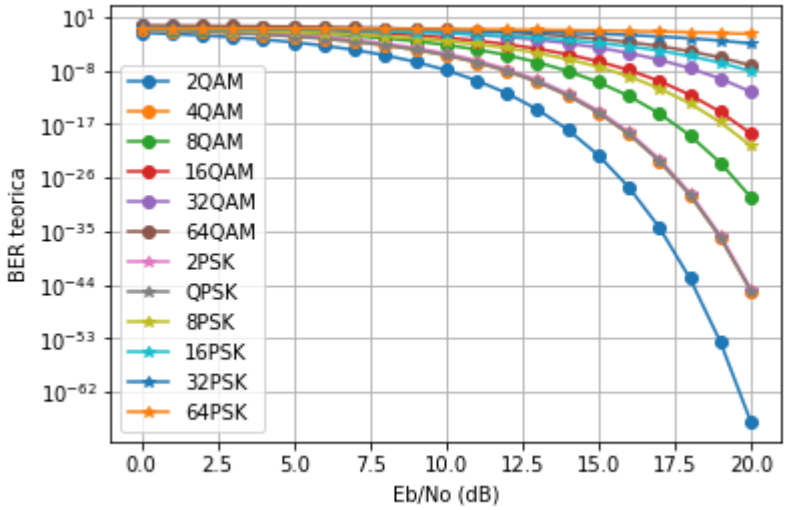
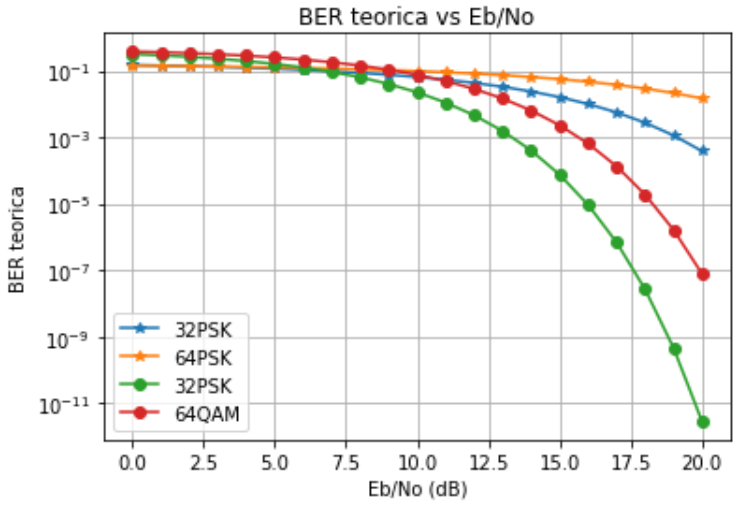


Figura 4-1. Representación de la BER teórica para distintas modulaciones M-arias (M-PSK y M-QAM)

De la gráfica de la figura 4-1 se pueden sacar varias conclusiones generales que se podrán aplicar a la simulación de un sistema digital de comunicaciones digitales, ya que se podrán obviar varios pasos y cálculos de probabilidad de error práctica para modulaciones que no llegan a la probabilidad de error mínima para que sea viable la comunicación.

- Conforme se van introduciendo más símbolos en la transmisión, la probabilidad de error se hace mayor para un mismo valor de E_b/N_o.
- La probabilidad de error es indirectamente proporcional al valor de la relación E_b/N_o, por lo que si esta crece la probabilidad de error cada vez es menor.
- Las modulaciones 2-PSK, QPSK y 4-QAM tienen la misma probabilidad para los distintos casos de E_b/N_o por lo que con evaluar una de ellas, se tendrá los resultados de las otras.
- Para el mismo valor de M, las modulaciones M-QAM tienen mejor probabilidad de error que las M-PSK para una misma E_b/N_o.

Respecto a la probabilidad de error, si se separa la gráfica en dos se puede observar mejor cuándo llegan las distintas curvas al valor objetivo de 10^{-6}.



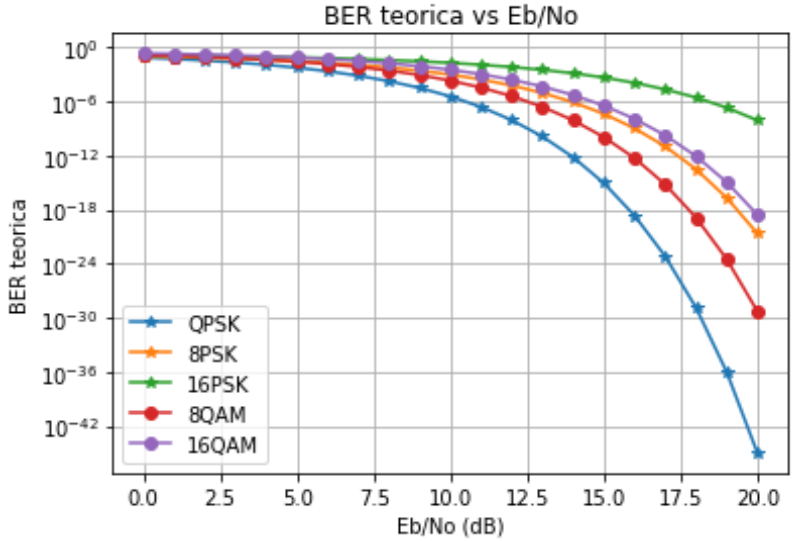


Figura 4-2. Representación ampliada en el eje vertical de la BER teórica para distintas modulaciones M-arias (M-PSK y M-QAM)

Tabla 4–1. Relación E_b/N_o a la que una modulación llega a una probabilidad de error de bit aproximada de 10^{-6}

Modulación	E_b/N_o (dB)
QPSK	10
8-PSK	13
8-QAM	12
16-PSK	18.5
16-QAM	14
32-PSK	23
32-QAM	19
64-PSK	28
64-QAM	32

4.2. Diseño de un sistema de radiocomunicación básico mediante Python

Para explicar el funcionamiento del sistema de radiocomunicación desarrollado en Python se tomará como ejemplo una modulación QPSK ya que es el mismo para distintos tipos de modulaciones, lo único que varía es la constelación resultante y la expresión de la probabilidad de error teórica.

4.2.1. Introducción de los datos a transmitir

Se crea un vector de datos binarios de longitud k·Ns para poder agrupar los bits de forma correcta en la modulación, donde k es el número de bits por símbolo y Ns el número de bits total. De esta forma, se puede asegurar que cada k bits haya 1 muestra sin que queden bits sueltos.

```
x_tx = np.random.randint(0, 2, k*Ns)
```

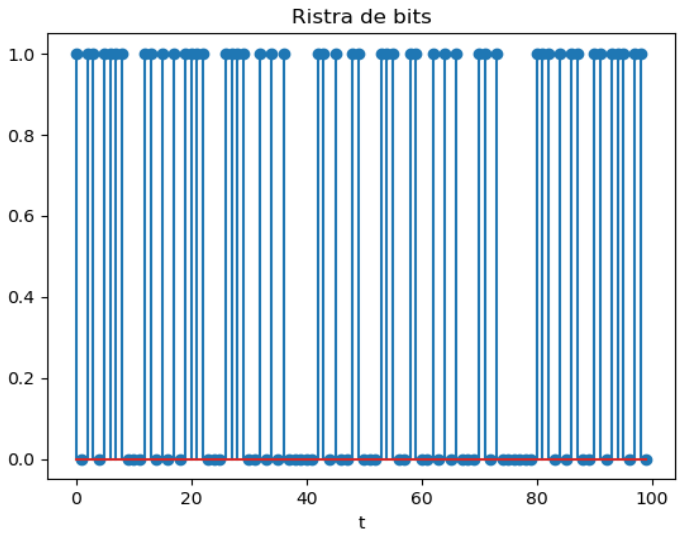


Figura 4-3. Representación de una parte del vector de datos binario, x_tx[1:100]

4.2.2. Creación de un objeto de las modulaciones PSK o QAM

Tras crear un objeto con modulación PSK o QAM, al modular la ristra de bits se obtiene la constelación de dicha modulación.

m_tx=mod1.modulate(x_tx)

Si se representa la constelación para M=4 en el caso de la PSK, se obtendrán 4 puntos a la misma distancia desde el centro (0,0) con variación de la fase con valores (π/2, π, 3π /4, 2π). En el caso de la QAM los puntos estarán equidistanciados a la misma distancia entre puntos consecutivos. La distancia que los separe estará asociada a la energía de la señal, por lo que para calcular la BER se tendrá en cuenta indirectamente, sabiendo que cuanto más separados estén los puntos, menor será la probabilidad de que se detecten los puntos de forma errónea. Los puntos tomarán los valores {+1, -1, j, -j}.

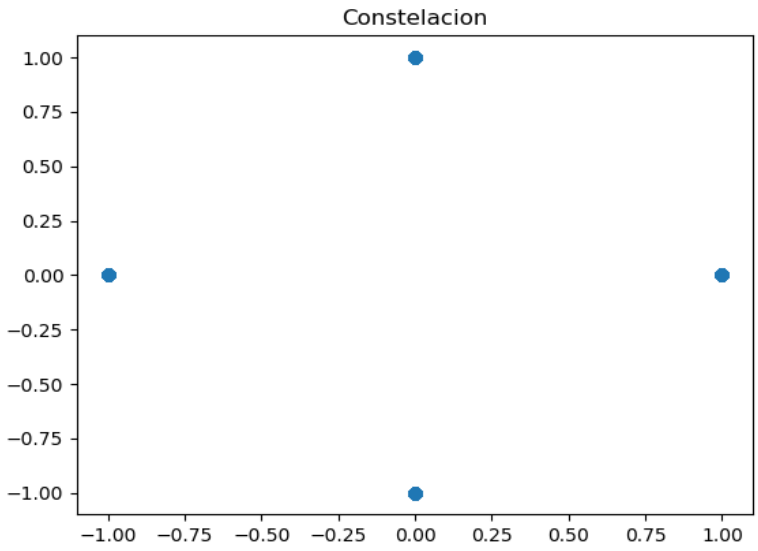


Figura 4-4.Constelación 4-PSK

Representando la parte en cuadratura de los símbolos se van a poder obtener 3 posibles valores {1, -1, 0}, lo que permitirá evaluar el comportamiento del sistema en el tiempo durante las distintas etapas, tanto en transmisión como en recepción.

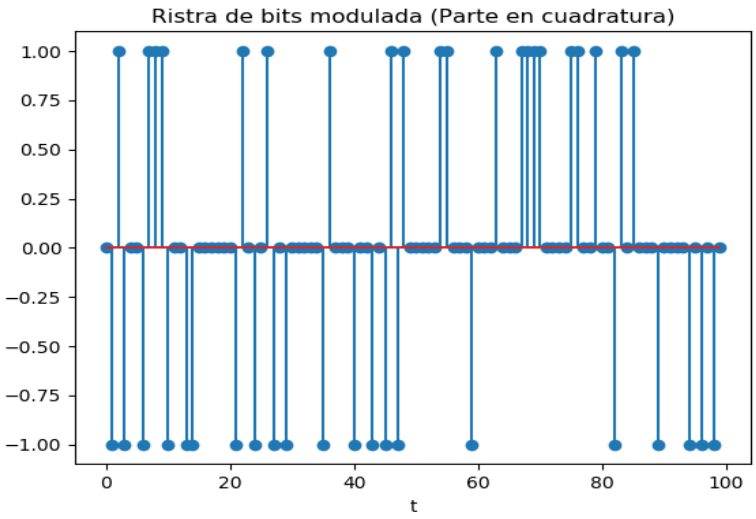


Figura 4-5. Parte en cuadratura de los símbolos

4.2.3. Sobremuestreo de la señal

El siguiente paso consiste en sobremuestrear la señal, añadiendo ceros tras cada valor de la señal modulada de forma que se obtendrán los símbolos que se quieren mandar cada nsamp, que son las muestras por símbolo. El sobremuestreo se realiza para disponer de un mayor nivel de detalle en la forma de onda después de que la señal pase por el filtro conformador de pulso.

Si se representa con saltos de nsamp, es decir, una muestra por símbolo, se obtienen que los símbolos transmitidos son solamente los que pertenecen a la constelación. En el tiempo quedará como la señal original, pero con ceros entre las muestras que son distintas de cero.

y_tx=upsample (m_tx, nsamp)

Las componentes en fase y cuadratura siguen formando la constelación original, sin embargo, entre muestra y muestra se han añadido ceros como se puede ver en la figura 4-6

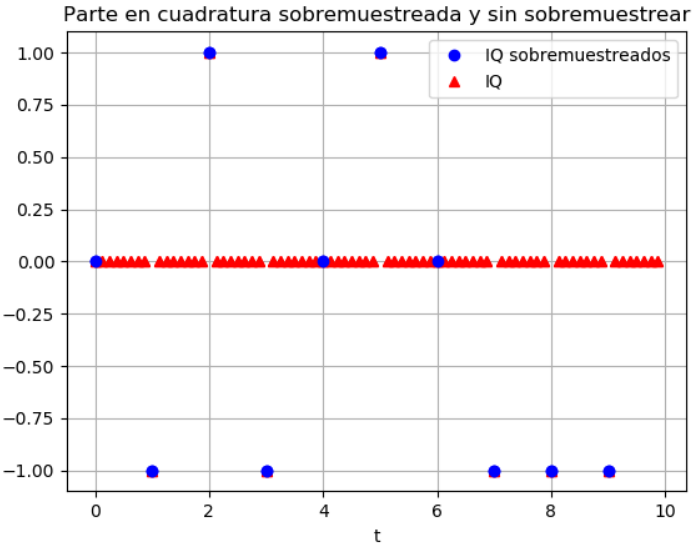


Figura 4-6. Parte en cuadratura sobremuestreada y sin Sobremuestrear

Las dos señales son la misma solo que al realizarse un sobremuestrado *zero-padding* producirá que cuando pase por el filtro raíz de coseno alzado la señal de salida tendrá una forma de onda más suave y

no tan abrupta como si no tuviese las muestras intermedias añadidas.

4.2.4. Filtrado de la señal en transmisión

En este proceso hay que tener en cuenta todos los parámetros que se vaya a utilizar en el filtro. Los valores más comunes del factor de *roll off* son 0.22 y 0.35, pero se añadirán también 0.5 y 0.8 para estudiar los distintos casos. Una vez especificados los parámetros se procederá a diseñar el filtro.

El filtro es raíz de coseno alzado y su diseño se realizará de la siguiente forma, teniendo en cuenta que nsamp es el número de muestras añadidas en el sobremuestreo entre los símbolos modulados y alpha el factor de *roll off*.

rcosalz = sqrt_rc_imp(nsamp, alpha)

La representación del filtro raíz de coseno alzado para los distintos factores de *roll off* se muestra en la figura 4-7.

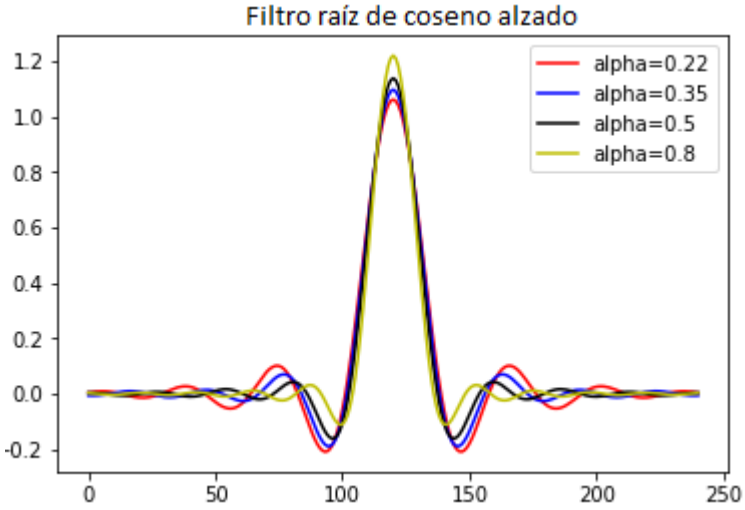


Figura 4-7. Representación de raíz de coseno alzado para distintos valores de alpha

Se ha diseñado el sistema con un filtro raíz de coseno alzado tanto en transmisor como en receptor por ser la opción más extendida en la implementación práctica de los sistemas de radiocomunicación. Como se ha visto de forma teórica en el capítulo 2, el filtrado se tiene que realizar siguiendo el criterio de Nyquist de la forma más aproximada posible ya que este es irrealizable por su no causalidad, por lo que se tendrá que hacer mediante la familia del coseno alzado. Ya que la respuesta global del sistema es la del coseno alzado, se divide su efecto tanto en transmisión como en recepción.

La finalidad de este filtro es darle forma a la señal que se va a transmitir, por lo que no varía la amplitud de los símbolos que se están transmitiendo.

r_tx=np.convolve(y_tx, rcosalz,mode='same')

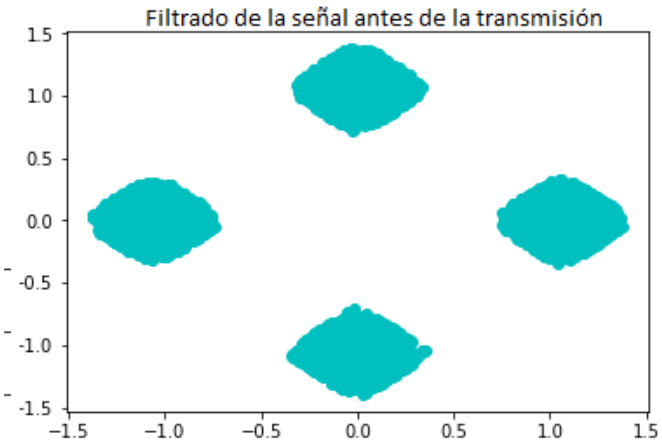


Figura 4-8. Representación de los símbolos a transmitir después de aplicar el filtro de transmisión

La señal se convoluciona con el filtro para prepararla para la transmisión, de modo que queda una señal continua en el tiempo:

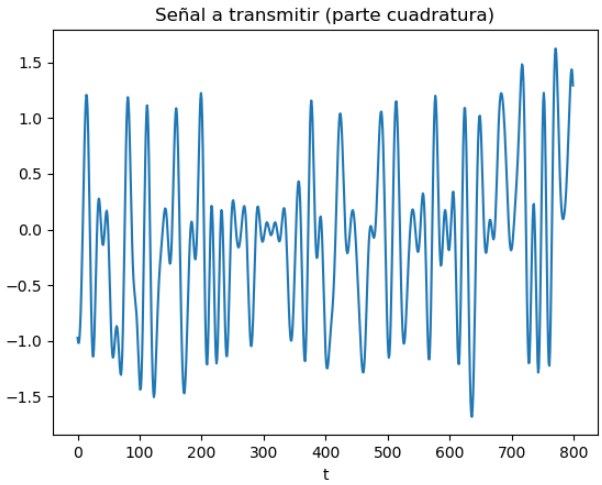


Figura 4-9. Señal en cuadratura preparada para la transmisión tras el filtro raíz de coseno alzado

4.2.5. Paso de la señal filtrada por un canal AWGN

Como la finalidad del proyecto es obtener la relación BER vs E_b/N_o, se tiene que aplicar el canal AWGN para los distintos valores de E_b/No mediante un bucle for e ir guardando los distintos resultados obtenidos de la BER en un vector. A menor relación E_b/N_o el ruido predominará en el sistema ya que es una de las fuentes de distorsión que se tiene en el sistema. En las figuras 4-10 y 4-11 se puede observar que cuanto menor será dicha relación, menor posibilidad hay de poder encontrar qué símbolo se había transmitido porque la distorsión es 10 veces peor para E_b/N_o= -5 dB que para E_b/N_o= 20 dB. Esto se debe a que, si un punto tiene mucho ruido, es posible que su región de admisibilidad esté más cerca de la de otros puntos que de la que debería. Aunque los puntos estén más juntos para E_b/N_o mayores, el ruido es más bajo por lo que la variación entre el símbolo transmitido y al que se le ha añadido ruido no será tan grande como para que no se reciba de forma correcta.

```
esno = ebno + 10*log(k,10)

w=cpx_AWGN(r_tx, esno, nsamp)
```

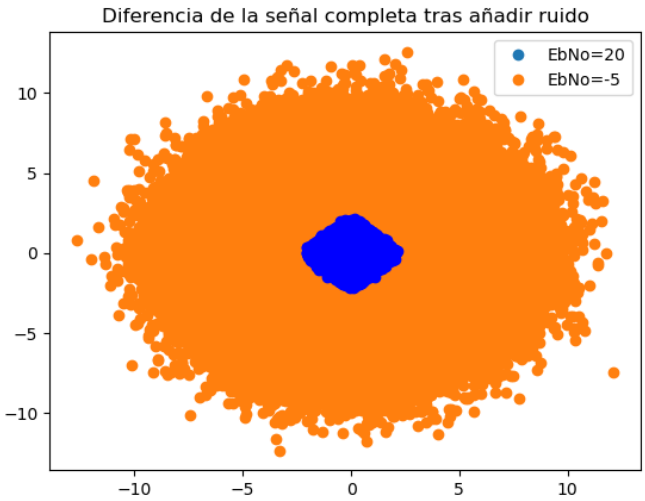


Figura 4-10. Representación de los símbolos después de pasar por el canal con mucho ruido y por otro con poco ruido. E_b/N_0 en dB.

Si se ve en el dominio del tiempo, se observa que con mucho ruido la señal varía mucho más que con poco ruido llegando a tal punto que puede ser que la comunicación sea imposible ya que no se puede distinguir bien el símbolo que se está transmitiendo.

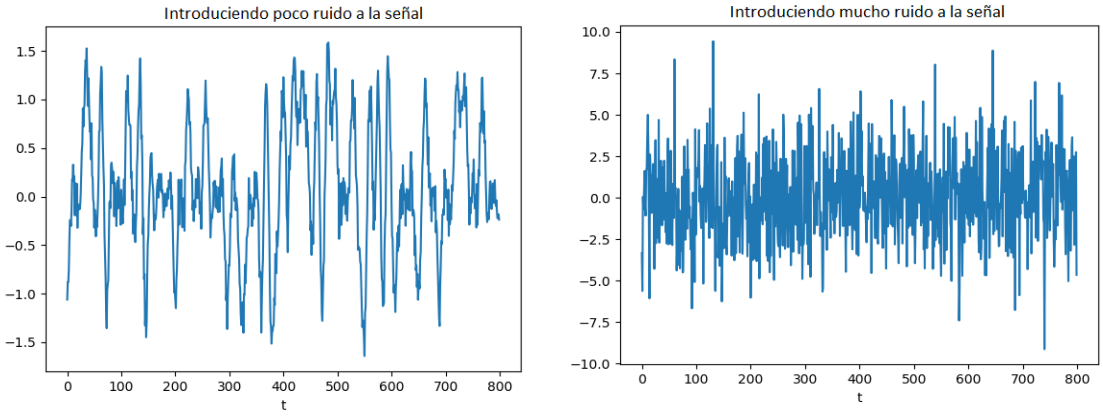


Figura 4-11. Influencia de la relación E_b/N_o en la transmisión. a) Introduciendo poco ruido. b) Introduciendo mucho ruido

4.2.6. Filtrado de la señal en recepción

Si el filtro de transmisión en un filtro raíz de coseno alzado, en recepción debe emplearse un filtro raíz de coseno alzado para que los filtros estén acoplados y eliminen la mayor cantidad de ISI posible.

```
y_rx=np.convolve(w,rcosalz,mode='same')
```

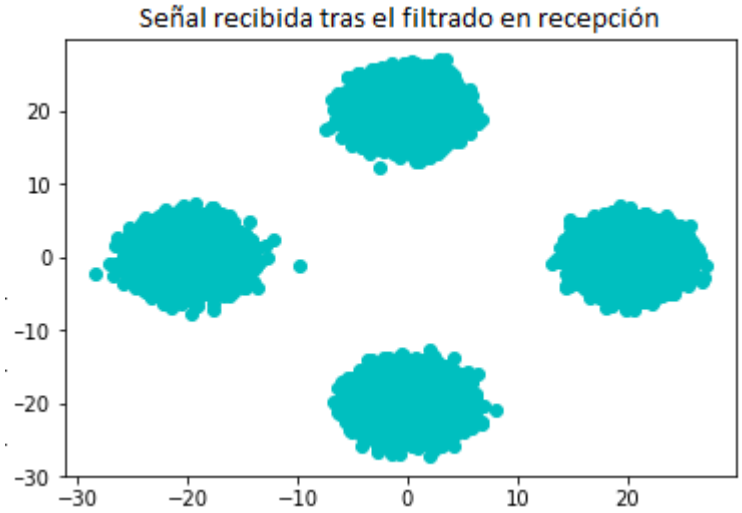


Figura 4-12. Representación de los símbolos recibidos después de aplicar el filtro de recepción

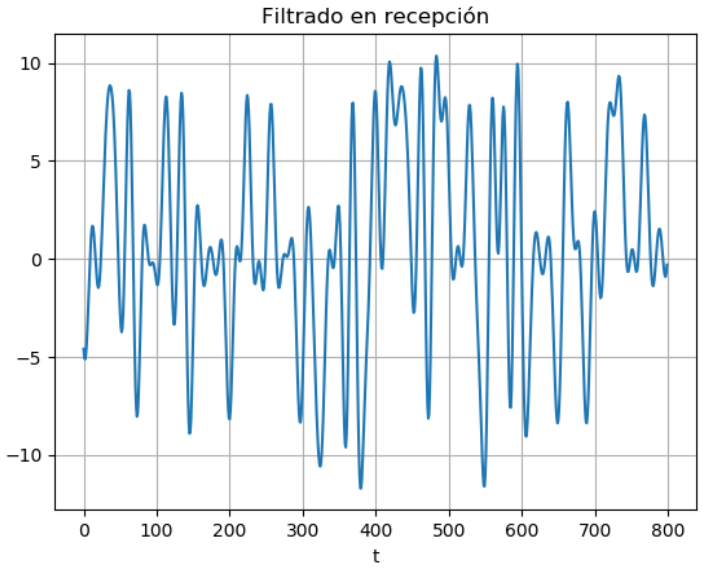


Figura 4-13. Representación de la señal en cuadratura después del filtrado en recepción.

Si se compara la señal transmitida con la recibida son iguales salvo por la excepción de la amplitud que es mayor en el caso de la recepción. Esto se debe a las sumas entre el ruido y la convolución del filtro con la señal recibida con el ruido.

Como se puede observar tanto en la representación de la constelación como en la de la señal, se ha eliminado la dispersión de los puntos existente no debida al ruido, ya que los filtros en transmisión y en recepción están acoplados.

4.2.7. Submuestreo de la señal

Una vez pasado por el filtro, se quitan todos las muestras en instantes intermedios del periodo de símbolo que se habían introducido en el sobremuestreo de la señal.

```
m_rx= downsample(y_rx,nsamp)
```

En el caso de la modulación PSK no haría falta, sin embargo, para la QAM es necesario normalizar en amplitud la señal tras el submuestreo del filtro comparándola con los símbolos transmitidos ya que, si no, el demodulador utilizado no realiza bien la detección debido a la forma en la que esta implementado.

```
m_rx=m_rx/np.linalg.norm(m_rx)*np.linalg.norm(m_tx)
```

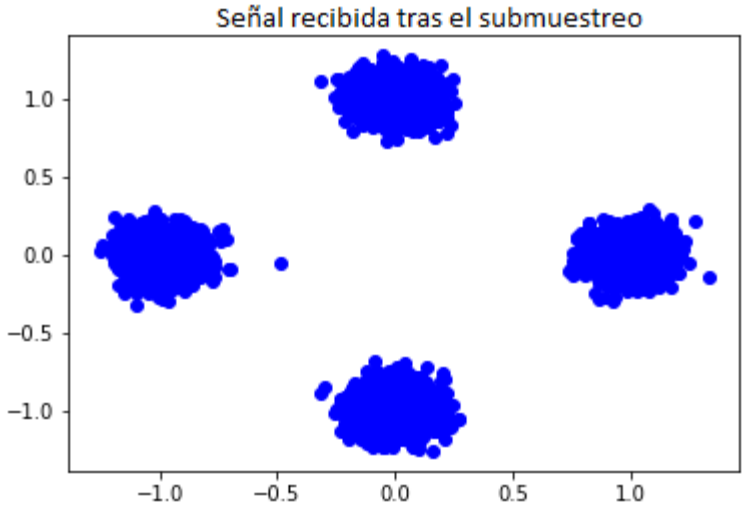


Figura 4-14. Representación de los símbolos recibidos tras submuestrear la señal

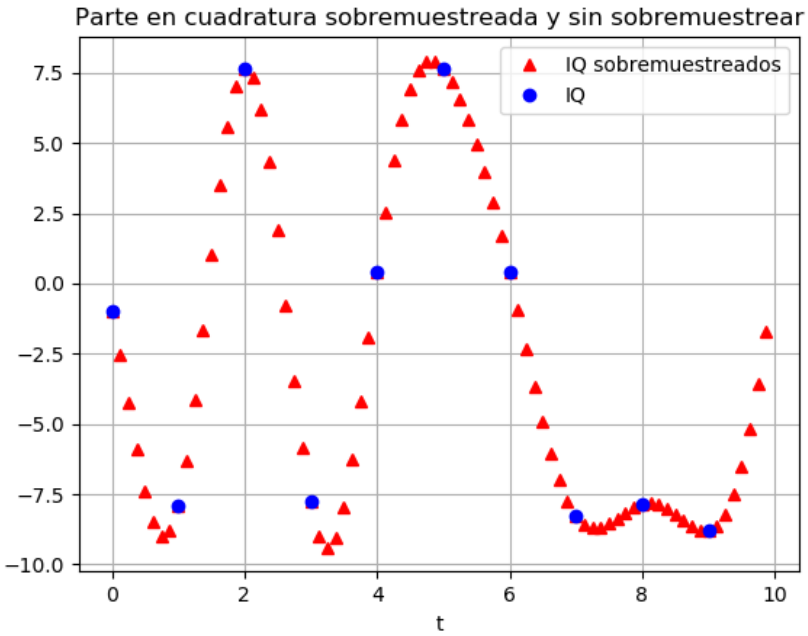


Figura 4-15. Representación de las muestras en cuadratura tras el submuestreo de la señal (IQ submuestreadas) y las muestras en cuadratura tras el filtro en recepción (IQ)

El submuestreo se realiza porque para demodular solo se necesita una muestra por símbolo.

4.2.8. Demodulación de la señal

Para obtener los bits transmitidos se demodula con una codificación de *hard-decision* en la que no se intenta hacer ninguna corrección, sino que se demodula tal y como lo ha recibido.

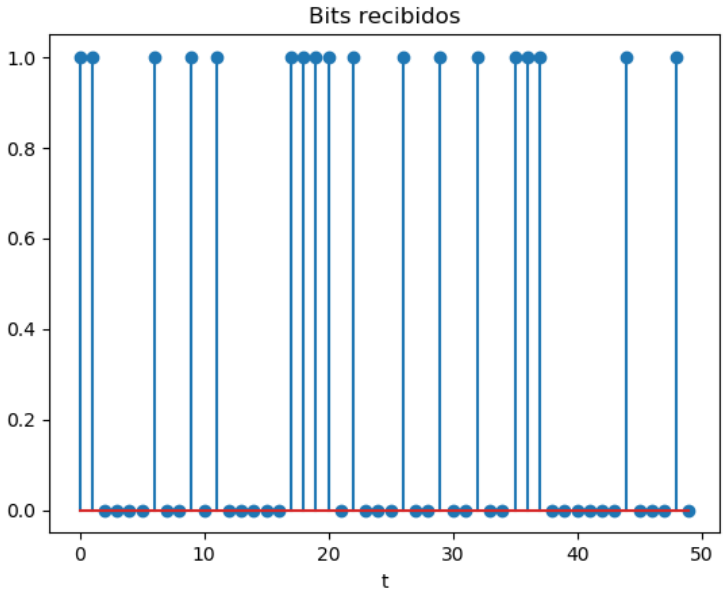


Figura 4-16. Bits recibidos

4.2.9. Cálculo de errores en la transmisión

Los errores en transmisión se calculan comparando los bits de transmisión y de recepción, siendo la tasa de error de bit el total de bits erróneos entre el total de bits.

```
for u,v in zip(x_tx,x_rx):
    if u!=v:
      error_sum+=1;
BER[loop] = float(error_sum)/float(total_sent)
```

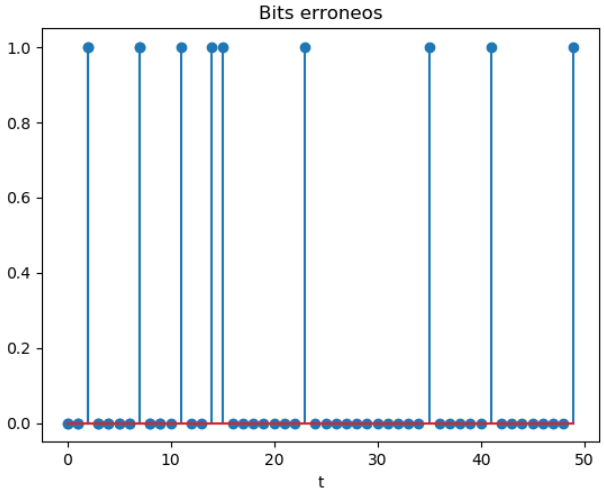


Figura 4-17. Bits erróneos: 1 si hay error, 0 si no lo hay.

En la figura 4-17 se ha cogido un intervalo de 0 a 50 del vector de bits erróneos, por lo que si solo fuesen los bits que se transmiten se tendría:

BER=número de errores / número total de bits= 9/50=0.18

De esta forma se calculará la gráfica de la BER respecto a la relación E_b/N_o del sistema. Como ya se ha visto, cuanto más pequeña sea mayor será la distorsión que se le aplique a la señal dando como resultado una peor calidad en la comunicación, por lo que la probabilidad de error será mayor que para E_b/N_o más grandes donde la distorsión será más pequeña y no un inconveniente para recibir la señal correctamente.

4.3. Diseño de un sistema de radiocomunicación utilizando OFDM mediante Python

Los parámetros que se han de tener en cuenta son:

- El número de subportadoras N, que depende del ancho de banda, la velocidad de los datos y la duración del tiempo útil. Será el número de puntos complejos que se procesan por la FFT/IFFT. En este proceso se tomarán valores Nfft=64 y Nfft=128 de referencia del estándar 802.11a y 802.11g, con valores respectivos de número de portadoras Ndatos=48 y Ndatos=128.
- El esquema de modulación: Si utiliza QAM o PSK sobre cada una de las subportadoras de OFDM, teniendo en cuenta las simulaciones vistas para cada una que servirán en la selección de M.

  mod1=QAMModem(M)
- Si tiene o no prefijo cíclico (CP) y cuál es su respuesta ante su presencia o ausencia. El CP sirve para hacer la señal periódica trabajando como banda de guarda entre los símbolos. Esto se hace copiando los últimos símbolos al principio y, aunque aumenta el número total de símbolos enviados, reduce la ISI ya que los símbolos que se pierden se pueden recuperar.

El esquema de una OFDM es más complejo que el esquema visto anteriormente de comunicaciones digitales en el que se incluyen los siguientes bloques:

- Convertidor serie/paralelo: los símbolos en serie se pasan a un formato en paralelo.
- Convertidor paralelo/serie: los símbolos en paralelo se pasan a un formato en serie.
- IFFT: genera N portadoras a diferentes frecuencias ortogonales, generando la modulación OFDM.
- FFT: transforma una señal periódica en el dominio del tiempo en su equivalente espectro de frecuencia, decodificando con el algoritmo adecuado para volver a hacer que los N subcanales vuelvan a ser uno.

  ofdm_tx_symbolos = OFDM_tx(m_tx, nd ,nfft,0,True,Ncp)

  donde m_tx es la señal modulada mediante PSK o QAM, nd es el número de portadoras que llevan datos, nfft es el número de subportadoras total, 0 indica que no hay portadoras piloto, True que tiene prefijo cíclico y que vale Ncp. Alpha es el factor de olvido que se añade en el estimador de canal. En este bloque se incluye la IFFT.

  m_rx,H = OFDM_rx(ofdm_rx_symbolos,nd,nfft,0,True,Ncp,alpha)

  En este paso se obtiene la respuesta impulsiva del canal H y los símbolos recibidos modulados mediante QAM o PSK. En este bloque se incluye la FFT.
- Codificador: convierte los bits en símbolos dependiendo de la modulación M-QAM o M-PSK

  Mod1=mod1.modulate(M)
- Decodificador: convierte los símbolos recibidos en un flujo de bits.

```
x_rx= mod1.demodulate(m_rx,'hard')
```

Para implementar el sistema se utilizará la combinación de las librerías de scikit_dsp y commpy. Se repetirán pasos que se han utilizado en el anterior caso de implementación de un sistema de radiocomunicación por lo que los pasos de datos a transmitir, modulación de bits (PSK o QAM), paso por el canal AWGN, demodulación de los bits y cálculo de la BER no se verán en este apartado ya que son idénticos al anterior

4.3.1. Implementación de OFDM en el transmisor

4.3.1.1. Conversión Serie-Paralelo

En este bloque se calcula el número de símbolos OFDM que será el número total de símbolos QAM/PSK entre el número total de subportadoras con datos .

Tras esto, se cambia la forma de la matriz que contiene a los símbolos y los reorganiza en N_OFDM filas y Ndatas columnas de modo que quedan repartidos los N_symb en N_OFDM·Ndatas celdas.

4.3.1.2. Buffer, transformada inversa rápida de Fourier e inclusión de prefijo cíclico

Los símbolos OFDM obtenidos en la conversión serie-paralelo van pasando un *buffer* fila por fila en el que se realiza la transformada inversa de Fourier. En el caso de que se vaya a incluir prefijo cíclico, se aprovecha el bucle for que va fila por fila en cada símbolo OFDM y concatena el tamaño de prefijo cíclico del final de la fila al principio del vector fila resultante.

4.3.1.3. Señal transmitida

La función OFDM_tx perteneciente a la librería sk_dsp.digitalcom se ha tenido que modificar en ciertas partes ya que no accedía bien a las funciones por identación, nombres de variables y acceso a funciones.

ofdm_tx_symbols= OFDM_tx(m_tx, nd, nfft, Np=0, cp, Ncp=0)

m_tx: símbolos a transmitir

nd: número de subportadoras de datos

nfft: número de subportadoras totales

Np: periodo de subportadoras. Este aspecto se obviará porque da problemas en la implementación que no se han conseguido solventar.

cp: True si tiene prefijo cíclico y False si no lo tiene.

Ncp: longitud del prefijo cíclico

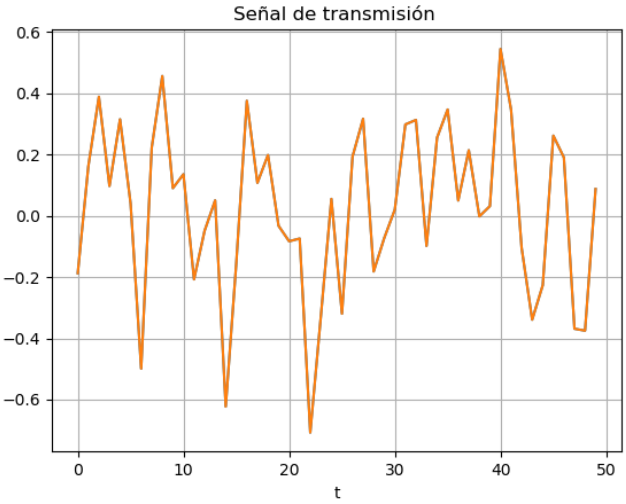


Figura 4-18. Señal en cuadratura de transmisión OFDM en el tiempo

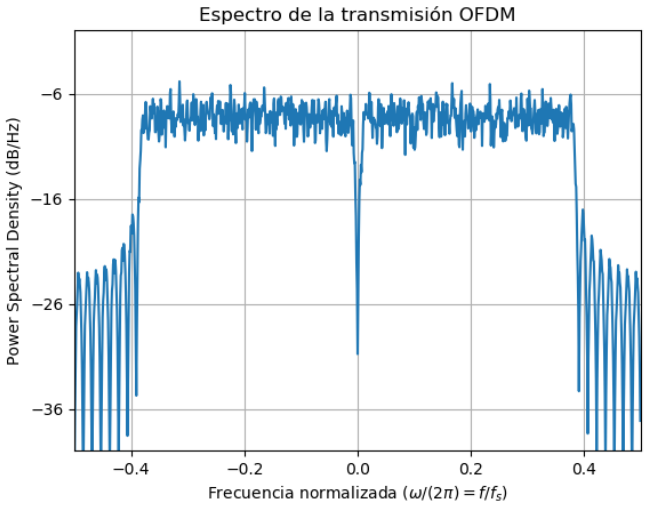


Figura 4-19. Señal de transmisión OFDM en el espectro de frecuencia

4.3.2. Implementación de OFDM en el receptor

4.1.1.1 Eliminación del prefijo cíclico, transformada rápida de Fourier y buffer

Primero se elimina el prefijo cíclico del *buffer* en el caso de que lo tuviese, después se guarda en el *buffer* y se hace la transformada rápida de Fourier a la fila a la que se le ha quitado el prefijo cíclico. Una vez se ha realizado en todas las filas, se pasa a la conversion paralelo-serie.

4.1.1.2 Conversión Paralelo-Serie

En este bloque se cambia de nuevo la señal que se ha obtenido tras la realización de la FFT introduciendo todos los símbolos demodulados OFDM con la forma de un vector de dimensiones Nsymb filas y Nfft columnas. Tras esto, con la orden hstack, se pone como un vector fila.

4.3.2.1. Señal recibida

La función OFDM_rx perteneciente a la librería sk_dsp.digitalcom, al igual que OFDM_tx, se ha tenido que modificar.

ofdm_rx_symbols= OFDM_rx(ofdm_rx_symbol, nd, nfft, Np=0, cp, Ncp=0, alpha=0.95, ht=None)

ofdm_rx_symbol: señal modulada con OFDM con ruido añadido

nd: número de subportadoras de datos

nfft: número de subportadoras totals

Np: periodo de subportadoras. Este aspecto se obviará porque da problemas en la implementación que no se han conseguido solventar.

cp: True si tiene prefijo cíclico y False si no lo tiene.

Ncp: longitud del prefijo cíclico

alpha: factor influyente en la estimación del canal

ht: respuesta impulsiva del canal. Para compararlo con el sistema implementado en el apartado 4.2. se tomará como que no hay.

Tras pasar por el ruido, el espectro en frecuencia sufre un cambio pero no tanto como en el tiempo. La desventaja de la E_b/N_o baja es que las subportadoras con datos están muy cerca de las bandas laterales ya que estas han aumentado mucho más con el ruido. Aún así, se sigue disintinguiendo la banda de señal en el espectro de la señal OFDM, cosa que no se puede decir si se representa la constelación en fase y cuadratura a no ser que la E_b/N_o sea alta.

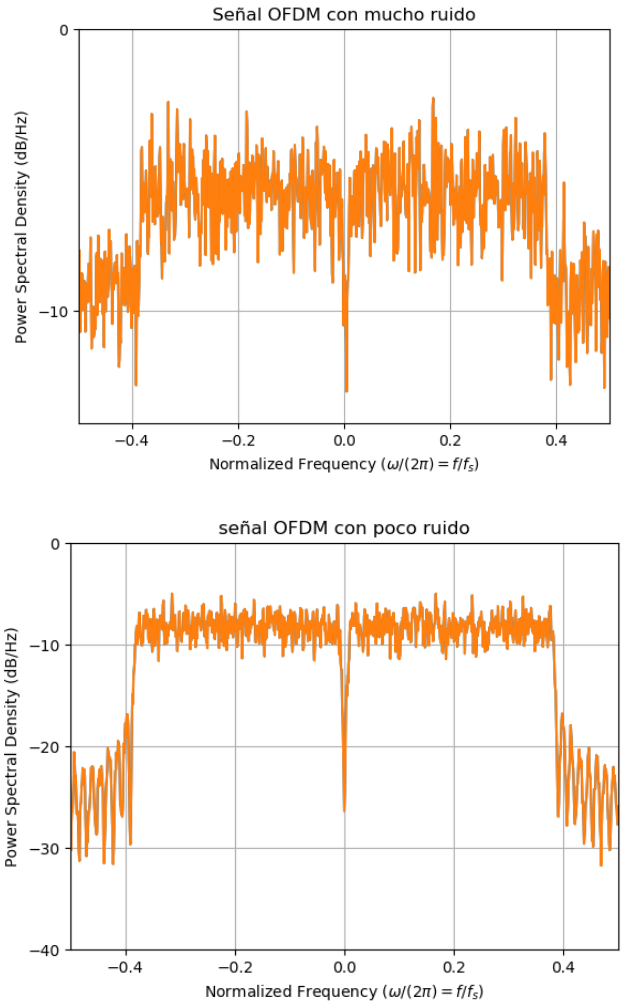


Figura 4-20. Influencia de la relación E_b/N_o en el espectro de la señal transmitida

La señal recibida variará en función de la relación E_b/N_o del sistema ya que el ruido puede afectar

gravemente en el sistema. Viendo varios ejemplos de E_b/N_o, los símbolos 16-QAM obtenidos se distinguen de forma muy distinta para los distintos valores.

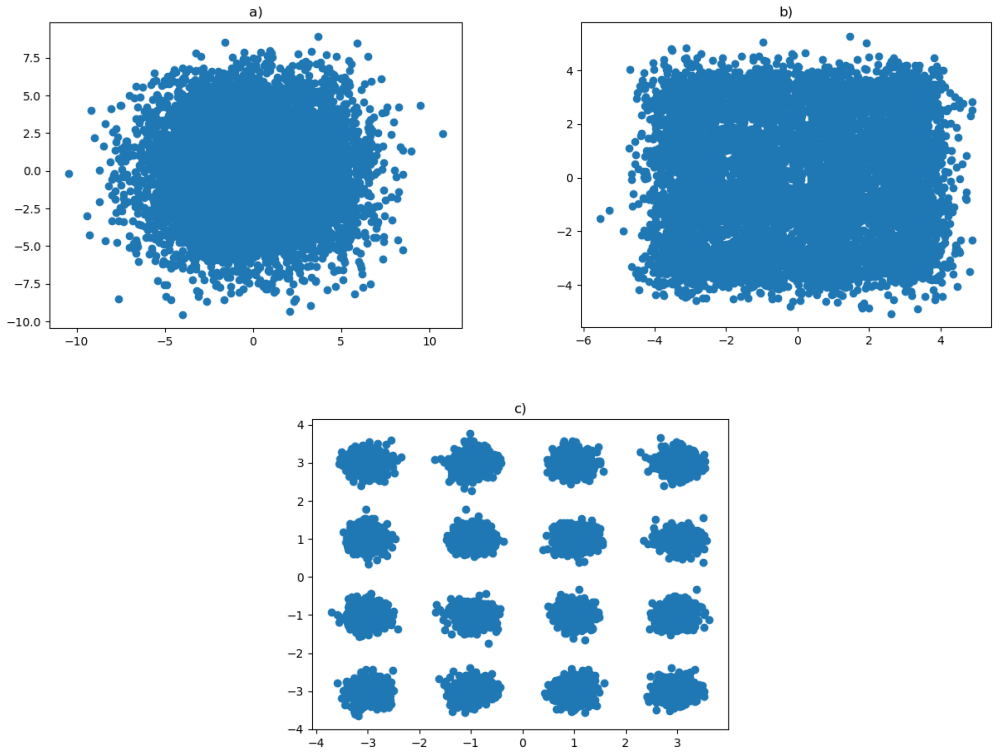


Figura 4-21. Influencia de la relación E_b/N_o en los símbolos recibidos. a) E_b/N_o =0 dB, b) E_b/N_o =10 dB, c) E_b/N_o =20 dB

4.4. Mediciones de la BER para modulaciones PSK

Para realizar los pasos mencionados en la introducción, la forma más comoda de hacer las medidas y representaciones es mediante una función que devuelva las gráficas que se están buscando de la BER.

```
def PSK_BER(M,Ns,EbNo_MIN,EbNo_MAX,color,alpha):
    """
    Esta función devuelve las gráficas teóricas y simuladas de la BER
    en un sistema de radiocomunicaciones para una modulacion MPSK
    Dependiendo de los parámetros que se les pase evaluará el sistema
    de una forma distinta
    M: Número de símbolos de la constelación
    Ns: Número de bits transmitidos
    EbNo_MIN,EbNo_MAX: intervalos de evaluación de la BER
    color: parámetro añadido por estética para representar varias gráficas
    alpha: factor de roll-off de los filtros de transmisión y recepción
    """
    ##Definicion del intervalo de EbNo
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
    ##Se crea un objeto te tipo PSK
    mod1=PSKModem(M)
    k=mod1.num_bits_symbol
    ##Se crearán variables vacías para ir añadiendo los valores
    Pe        = np.empty(np.shape(Eb_No_lin))
    BER       = np.empty(np.shape(Eb_No_lin))
```

```
##Parámetros para el diseño del filtro
Ts=1;
Fs=8; #Cuanto más suba Fs, más se acercará la BER a la gráfica teórica
nsamp=Fs*Ts; #Muestras por simbolo

##Diseño del filtro
filtro= sqrt_rc_imp(nsamp,alpha)
##Crear el vector de datos binarios
x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
##Aplicar la modulacion PSK a los bits
m_tx=mod1.modulate(x_tx)

##Sobremuestrear y filtrar los datos de entrada
y_tx=upsample(m_tx,nsamp)
r_tx=np.convolve(y_tx,filtro,mode='same')
##Pasar la señal por un canal AWGN
loop=0;
for ebno in Eb_No_dB:
    error_sum=0;
    esno = ebno + 10*log(k,10)
    w=cpx_AWGN(r_tx,esno,nsamp)
    ##Filtrar y submuestrear
    y_rx=np.convolve(w,filtro,mode='same')
    ##Submuestrear la señal recibida
    m_rx= downsample(y_rx,nsamp)
   ##Demodular la señal recibida
    x_rx= mod1.demodulate(m_rx,'hard')
    total_sent=len(x_rx)
    ##Se comprueban los errores comparando la transmisión con la recepción
    for u,v in zip(x_tx,x_rx):
        if u!=v:
            error_sum+=1;
    ##Cálculo de la Tasa de Error por Bit
    BER[loop] = float(error_sum)/float(total_sent)
    loop += 1

loop=0;
##Cálculo de la probabilidad de error teórica
for en in Eb_No_lin:
    Pe[loop]  = (2/log(M,2))*Q_fctn(sqrt(2*log(M,2)*en)*sin(pi/M))
    loop+=1
plt.ylim(10**-7)
#plt.semilogy(Eb_No_dB, Pe,color,linewidth=2)
plt.semilogy(Eb_No_dB, BER,color)
```

En el estudio del comportamiento de la PSK en un sistema de comunicaciones digitales, se comenzará estudiando las modulaciones QPSK, 8-PSK, 16-PSK, 32-PSK y 64-PSK para el caso ideal, es decir, el teórico. De esta forma, si no cumple la restricción de probabilidad de error mínima para que la comunicación se acerce a la calidad objetivo, se obviará. Así se optimizará el tiempo de simulación debido a su alto peso computacional ya que no aportan ninguna información aparte de que no sean adecuados para el sistema.

4.5. Diseño del sistema de comunicaciones digitales para modulación M-PSK

4.5.1. Sistema de comunicaciones digitales con filtro raíz de coseno alzado y factor de roll off de 0.22

Con el siguiente trozo de código conseguimos los resultados para el caso tanto teórico como práctico

del sistema utilizando QPSK, 8-PSK, 16-PSK, 32-PSK, 64-PSK con un filtro raíz de coseno alzado y factor de *roll off* de 0.22.

```
#PRUEBA 1: Distintos valores de M, filtro raiz de coseno alzado, alpha=0.22
plt.figure(1)
PSK_BER(4,1000000,-5,15,'-b',1,0.22) #QPSK
PSK_BER(8,1000000,-5,15,'-r',1,0.22) #8PSK
PSK_BER(16,100000,-5,15,'-g',1,0.22) #16PSK
PSK_BER(32,100000,-5,15,'-c',1,0.22) #32PSK
PSK_BER(64,100000,-5,15,'-k',1,0.22) #64PSK
plt.grid(True)
plt.legend(('QPSK teorica','QPSK simulada','8PSK teorica','8PSK simulada',
            '16PSK teorica','16PSK simulada',
            '32PSK teorica','32PSK simulada',
            '64PSK teorica','64PSK simulada'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

De la figura 4-22 se llega a la conclusión de que los valores deseados de la BER teórica menores que 10^{-6} solo se pueden conseguir para modulaciones QPSK y 8-PSK dependiendo de los valores de la E_b/N_o razonables y para la modulación 16-PSK, pero se necesitarían valores mayores de E_b/N_o. El estudio para los distintos factores de *roll off* se va a realizar exclusivamente para M=4 y M=8, ya que son los más utilizados para modulaciones PSK. Además de esto, se puede observar que para una menor E_b/N_o con la modulación QPSK se consigue mejor probabilidad de error con la desventaja de poder enviar menos símbolos en la transmisión y si se desea enviar más símbolos, se tendrá la desventaja de que la probabilidad de error sea mayor.

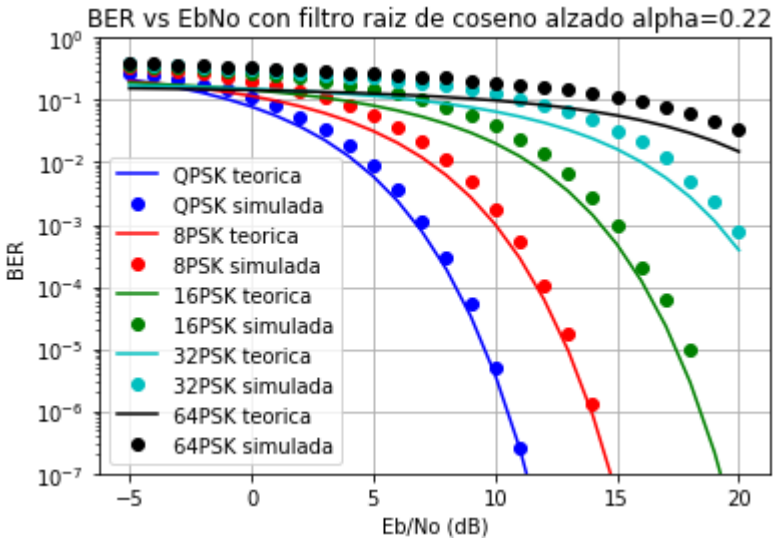


Figura 4-22. Representación QPSK, 8-PSK, 16-PSK, 32-PSK, 64-PSK con un factor de *roll off* de 0.22

Las simulaciones son muy aproximadas a las curvas teóricas por lo que se puede concluir que, para PSK, el sistema está correctamente implementado.

4.5.2. Tasa de error de bit para un sistema de comunicaciones con distintos factores de roll off

```
#%% PRUEBA 2: Distintos valores de alpha para M=4
plt.figure(2)
plt.title('BER vs EbNo con distintos valores de roll-off')
BER_teorica_PSK(4,'c')
PSK_BER(4,2000000,-5,20,'*b',0.22)
PSK_BER(4,2000000,-5,20,'or',0.35)
PSK_BER(4,2000000,-5,20,'vg',0.5)
PSK_BER(4,2000000,-5,20,'pk',0.8)
plt.grid(True)
plt.legend(('QPSK teorica','QPSK alpha=0.22',
            'QPSK alpha=0.35','QPSK alpha=0.5',
            'QPSK alpha=0.8'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

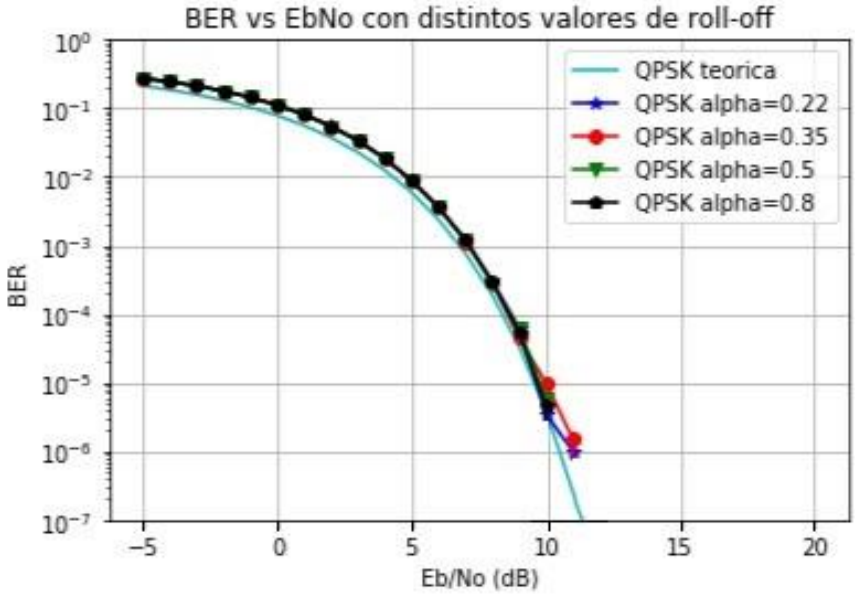


Figura 4-23. BER vs E_b/N_o para un sistema de comunicaciones con factores de *roll off* distintos

Si el sistema varía dependiendo del factor de *roll off* que se introduzca para el filtro raíz de coseno alzado, las variaciones son mínimas por lo que lo óptimo sería utilizar el factor de *roll off* más pequeño para que este ocupe el menor ancho de banda posible.

4.6. Diseño del sistema de comunicaciones digitales para modulación M-QAM

De la misma forma que para una modulación PSK, se va a evaluar un sistema de comunicaciones digitales siguiendo los pasos del apartado 4.1.

```
def QAM_BER(M,Ns,EbNo_MIN,EbNo_MAX,color,alpha):
    """
    Esta función devuelve las gráficas teóricas y simuladas de la BER
    en un sistema de radiocomunicaciones para una modulacion MQAM
    Dependiendo de los parámetros que se les pase evaluará el sistema
    de una forma distinta
    M: Número de símbolos de la constelación
    Ns: Número de bits transmitidos
    EbNo_MIN,EbNo_MAX: intervalos de evaluación de la BER
    color: parámetro añadido por estética para representar varias gráficas
    alpha: factor de roll-off de los filtros de transmisión y recepción
    """
    ##Definicion del intervalo de EbNo
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
    ##Se crea un objeto te tipo QAM
    mod1=QAMModem(M)
    k=mod1.num_bits_symbol
    ##Se crearán variables vacías para ir añadiendo informacion
    Pe        = np.empty(np.shape(Eb_No_lin))
    BER       = np.empty(np.shape(Eb_No_lin))

    ##Parñametros para el diseño del filtro
    Ts=1;
    Fs=8; #Cuanto más suba Fs, más se acercará la BER a la gráfica teórica
    nsamp=Fs*Ts; #Muestras por simbolo

    ##Diseño del filtro
    filtro= sqrt_rc_imp(nsamp,alpha)
    ##Crear el vector de datos binarios
    x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
    ##Aplicar la modulacion QAM a los bits
    m_tx=mod1.modulate(x_tx)
    ##Sobremuestrear y filtrar los datos de entrada
    y_tx=upsample(m_tx,nsamp)
    r_tx=np.convolve(y_tx,filtro,mode='same')
    ##Pasar la señal por un canal AWGN
    loop=0;
    for ebno in Eb_No_dB:
        error_sum=0;
        esno = ebno + 10*log(k,10)
        w=cpx_AWGN(r_tx,esno,nsamp)
        ##Filtrar la señal tras pasarla por el canal
        y_rx=np.convolve(w,filtro,mode='same')
        ##Submuestrear la señal recibida y linealizarla
        m_rx= downsample(y_rx,nsamp)
        m_rx=m_rx/np.linalg.norm(m_rx)*np.linalg.norm(m_tx)
        ##Demodular la señal recibida
        x_rx= mod1.demodulate(m_rx,'hard')
        total_sent=len(x_rx)
        #S#e comprueban los errores comparando la transmisión con la recepción
        for u,v in zip(x_tx,x_rx):
            if u!=v:
                error_sum+=1;
        ##Cálculo de la Tasa de Error por Bit
        BER[loop] = float(error_sum)/float(total_sent)
```

```
##Cálculo de la probabilidad de error teórica
loop=0;
for en in Eb_No_lin:
    term1=(1-(1/sqrt(M)))*erfc(sqrt(3*en*k/(2*(M-1))))
    term2=(1-(0.5*(1-(1/sqrt(M)))))*erfc(sqrt(3*en*k/(2*(M-1)))))
    Pe[loop] = term1*term2
    loop+=1
plt.ylim(10**-7)
#plt.semilogy(Eb_No_dB, Pe,color,linewidth=2)
plt.semilogy(Eb_No_dB, BER,color)
```

Todas las representaciones que se hagan a partir de este punto se representarán hasta una relación E_b/N_o para conseguir una probabilidad de error de 10^{-6} o aproximada para todos los valores de M que se representen. En el estudio previo se tuvo en cuenta que las representaciones de la QPSK/4-QAM y la 8-PSK eran parecidos a los de la 4-QAM y 8-QAM, pero a partir de M=16 esta modulación, aunque sea más difícil de implementar, consigue una mejor probabilidad de error en el sistema, por lo que se tomarán valores de M=16, M=32 y M=64.

4.6.1. Sistema de comunicaciones digitales con factor de roll off de 0.22

```
#%%PRUEBA 1: Distintos valores de M  alpha=0.22
plt.figure(1)
plt.title('BER vs EbNo con filtro raiz de coseno alzado alpha=0.22 ')
BER_teorica_QAM(4,'b')
QAM_BER(4,2000000,-5,20,'ob',0.22) #4QAM
BER_teorica_QAM(8,'r')
QAM_BER(8,2000000,-5,20,'or',0.22) #8QAM
BER_teorica_QAM(16,'g')
QAM_BER(16,200000,-5,20,'og',0.22) #16QAM
BER_teorica_QAM(32,'c')
QAM_BER(32,100000,-5,20,'oc',0.22) #32QAM
BER_teorica_QAM(64,'k')
QAM_BER(64,100000,-5,20,'ok',0.22) #64QAM
plt.grid(True)
plt.legend(('4QAM teorica','4QAM simulada',
            '8QAM teorica','8QAM simulada',
            '16QAM teorica','16QAM simulada',
            '32QAM teorica','32QAM simulada',
            '64QAM teorica','64QAM simulada'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

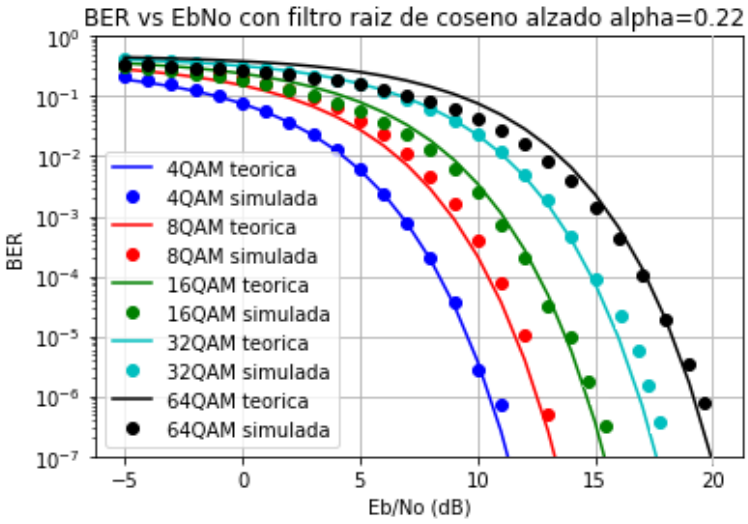


Figura 4-24. Representación 16-QAM, 32-QAM, 64-QAM y factor de *roll off* de 0.22

Si se desea obtener una probabilidad de 10^{-6}, para todas las M se consigue este valor a costa de incrementar la energía de bit para que los bits estén lo suficientemente separados como para modular/demodular todos los símbolos pertenecientes a la constelación. A mayor energía, mayor distancia entre símbolos y por ende menor probabilidad de error, de que se detecte como otro símbolo.

4.6.2. Tasa de error de bit para un sistema de comunicaciones con distintos factores de roll off

```
#%% Prueba 2. Para un mismo valor de M, distintos valores de roll-off
plt.figure(6)
plt.title('BER vs EbNo con distintos valores de roll-off')
BER_teorica_QAM(16,'b')
QAM_BER(16,100000,-5,15,'*g',0.22)
QAM_BER(16,10000,-5,15,'oc',0.35)
QAM_BER(16,10000,-5,15,'pk',0.5)
QAM_BER(16,10000,-5,20,'vr',0.8)
plt.ylim(10^-7)
plt.grid(True)
plt.legend(('16QAM teorica','16QAM alpha=0.22',
            '16QAM alpha=0.35','16QAM alpha=0.5','16QAM alpha=0.8'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

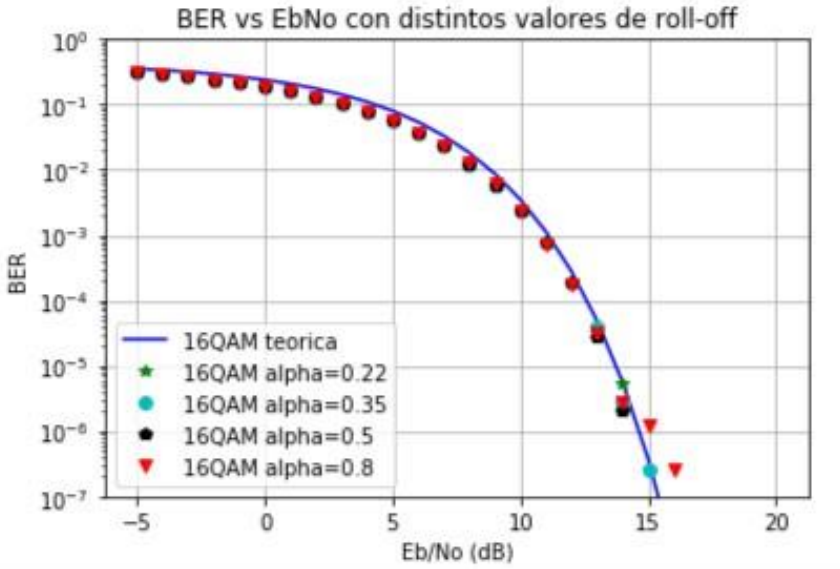


Figura 4-25. BER vs Eb/No para distintos factores de *roll off*

4.7. Comparaciones de simulaciones QAM Y PSK

Para valores pequeños de M como M=4 o M=8 conviene utilizar una modulación PSK por su simplicidad en la implementación ya que la 4-QAM simulada y la 4-PSK dan los mismos resultados y la 8-PSK una mejor tasa de error de bit que la 8-QAM para los mismos valores de E_b/N_o. A partir de M=16 la probabilidad de error de la PSK aumenta considerablemente más que la de QAM, por lo que se utiliza esta modulación para ello. Para M=16 todavía se pueden conseguir valores de 10^{-6} para una E_b/N_o de aproximadamente 20 dB, pero si seguimos con las potencias de dos siguientes se necesitaría una relación E_b/N_o demasiado grande que, comparándola con la implementación de QAM para obtener los mismos valores de BER, sería ineficiente y una pérdida de recursos innecesaria.

Tras el estudio previo realizado al comienzo del capítulo se obtiene un sistema de radiocomunicación con modulaciones QAM y PSK y filtrado en ambos extremos con raíz de coseno alzado con las

siguientes combinaciones QPSK, 8-PSK, 16-QAM, 32-QAM y 64-QAM donde los factores de *roll off* más eficientes son 0.22 y 0.35. Normalmente son las características que se suelen implementar en los sistemas de radiocomunicación reales.

```
#%% Prueba 1.Comparación entre PSK y QAM
plt.figure(6)
plt.title('BER vs EbNo con distintos valores de roll-off')
PSK_BER(4,1000000,-5,20,'-ob',0.22)
QAM_BER(4,1000000,-5,20,'-*b',0.22)
PSK_BER(8,1000000,-5,20,'-og',0.22)
QAM_BER(8,1000000,-5,20,'-*g',0.22)
PSK_BER(16,1000000,-5,20,'-or',0.22)
QAM_BER(16,1000000,-5,20,'-*r',0.22)
PSK_BER(32,1000000,-5,20,'-ok',0.22)
QAM_BER(32,1000000,-5,20,'-*k',0.22)
PSK_BER(64,1000000,-5,20,'-om',0.22)
QAM_BER(64,1000000,-5,20,'-*m',0.22)
plt.grid(True)
plt.legend(('QPSK','4QAM','8PSK','8QAM',
            '16PSK','16QAM','32PSK','32QAM',
            '64PSK','64QAM'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

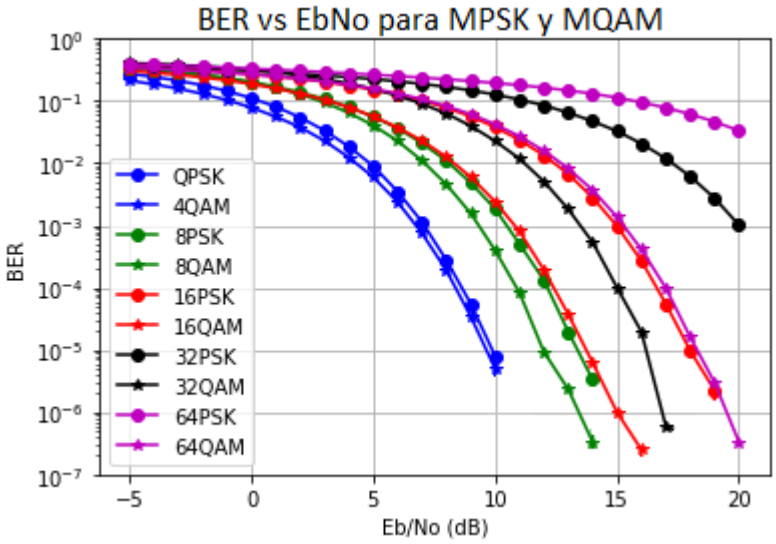


Figura 4-26. BER vs E_b/N_o para distintas modulaciones PSK y QAM

4.8. Mediciones de OFDM utilizando QPSK, 16-QAM y 64-QAM

Un sistema de radiocomunicación modulado mediante OFDM se diferencia de un sistema de radiocomunicaciones convencional con filtrado con un filtro de la familia del coseno alzado en los pasos de modulación del pulso, es decir, la modulación PSK o QAM se realiza para ambos de forma igual, sin embargo, su transmisión se modula de forma diferente. En este caso, se separa la ristra de símbolos completa en NFFT, número total de subportadoras, en las que los símbolos están en los NDATOS, número de subportadoras totales con datos. Se implementarán los estándares 802.11a y g que tienen un NFFT=64 y NDATA=48 y NFFT=128.

```
def ofdm_BER(M,EbNo_MIN,EbNo_MAX,color,nfft,nd,Ncp=0,modo=1):
    """
    M indica el número de símbolos que hay en la constelación
    EbNo_MIN,EbNo_MAX son los intervalos de evaluación de la BER
    color: para la representación de distintas gráficas
    nfft: número total de subportadoras
    nd: número de portadoras con datos
    Ncp: longitud de prefijo cíclico
    modo=1 -->QAM(por defecto)
    modo=0 -->PSK(para 4PSK)
    """
    #Definición del intervalo de EbNo
    Ns=3000000
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
    #Se crearán variables vacias para ir añadiendo los resultados
    Pe= np.empty(np.shape(Eb_No_lin))
    BER= np.empty(np.shape(Eb_No_lin))
     ##Se crea un objeto te tipo QAM
    if modo==0:
        mod1=PSKModem(M)
    else:
        mod1=QAMModem(M)
    k=mod1.num_bits_symbol
     ##Crear el vector de datos binarios
    x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
    total_sent=len(x_tx) #Se utilizará para medir la BER
    ##Aplicar la modulacion QAM a los bits
    m_tx=mod1.modulate(x_tx)
    #Señal modulada mediante OFDM a transmitir
    ofdm_tx_symbols= OFDM_tx(m_tx,nd,nfft,0,True,Ncp)
    ##Pasar la señal por un canal AWGN
    loop=0;
    for ebno in Eb_No_dB:
        error_sum=0;
        esno = ebno + 10*log(k,10)
        ofdm_rx_symbols=cpx_AWGN(ofdm_tx_symbols,esno,nfft//nd)

        ##Submuestrear la señal recibida y linealizarla
        m_rx= OFDM_rx(ofdm_rx_symbols,nd,nfft,0,estado,Ncp,0.95)
        ##Demodular la señal recibida
        x_rx= mod1.demodulate(m_rx,'hard')
        #S#e comprueban los errores comparando la transmisión con la recepción
        for u,v in zip(x_tx,x_rx):
            if u!=v:
                error_sum+=1;
        ##Cálculo de la Tasa de Error por Bit
        BER[loop] = float(error_sum)/float(total_sent)
        loop += 1

    ##Cálculo de la probabilidad de error teórica
    loop=0;
    for en in Eb_No_lin:
        term1=(1-(1/sqrt(M)))*erfc(sqrt(3*en*k/(2*(M-1))))
        term2=(1-(0.5*(1-(1/sqrt(M)))))*erfc(sqrt(3*en*k/(2*(M-1)))))
        Pe[loop] = term1*term2
        loop+=1
    plt.ylim(10**-7)
    #plt.semilogy(Eb_No_dB, Pe,color,linewidth=2)
    plt.semilogy(Eb_No_dB, BER,'-o'+color)
```

4.8.1. Esquema OFDM sin prefijo cíclico

4.8.1.1. Nfft=64 y Natos=48

```
#%%Prueba 1. OFDM con nfft=64 y ndatos=48 para distintas modulaciones sin CP
plt.figure(1)
plt.title('BER vs EbNo para modulacion OFDM con variacion de M y sin CP')
BER_teorica_PSK(4,'b')
BER_teorica_QAM(16,'r')
BER_teorica_QAM(64,'g')
ofdm_BER(4,-5,20,'b',64,48,Ncp=0,modo=0)
ofdm_BER(16,-5,20,'r',64,48,Ncp=0,modo=1)
ofdm_BER(64,-5,20,'g',64,48,Ncp=0,modo=1)
plt.grid(True)
plt.legend(('QPSK teorica','16QAM teorica',
            '64QAM teorica','OFDM/QPSK simulada',
            'OFDM/16QAM simulada','OFDM/64QAM simulada'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

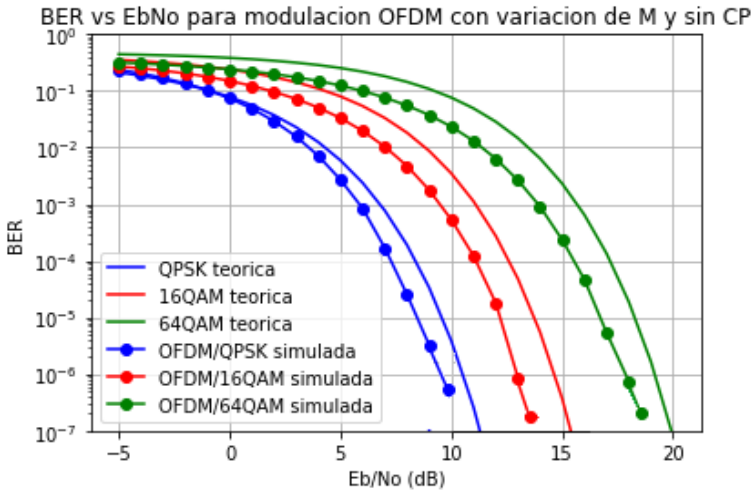


Figura 4-27. BER vs E_b/N_o para modulación OFDM y variación de M

Para las distintas modulaciones se consigue una mejora de la tasa de error por bit comparada con la tasa de bit teórica que es a la máxima que se podía llegar con una modulación con QAM o PSK simple.

4.8.1.2. Comparación en la utilización de distinto número de subportadoras para una M concreta

```
#%% Prueba 5. OFDM
plt.figure(1)
plt.title('BER vs EbNo con variacion de nfft')
BER_teorica_QAM(16,'b')
ofdm_BER(16,-5,20,'^k',16,10,True,Ncp=10,modo=1)
ofdm_BER(16,-5,20,'*r',64,48,False,Ncp=10,modo=1)
ofdm_BER(16,-5,20,'pr',128,108,True,Ncp=10,modo=1)
ofdm_BER(16,-5,20,'sg',1024,980,True,Ncp=10,modo=1)

plt.grid(True)
plt.legend(('16QAM teorica','NFFT=16','NFFT=64','NFFT=128','NFFT=1024'))
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

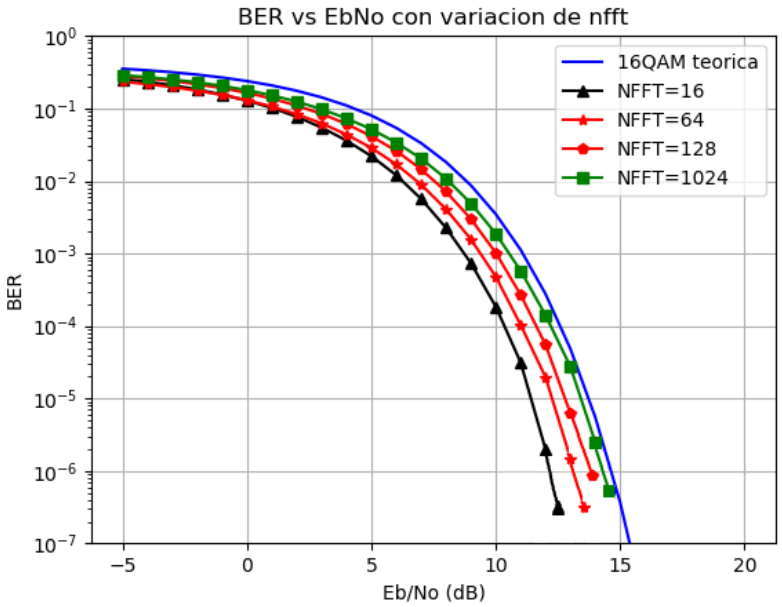


Figura 4-28. BER vs E_b/N_o para modulación OFDM con tamaño de la FFT variable

Tras realizar la simulación, se observa que para el diseño de este sistema para distintos números de subportadoras totales, conforme se van aumentando, la probabilidad de error aumenta.

4.8.1.3. Comparación en la utilización de Nfft fija y Ndata variable

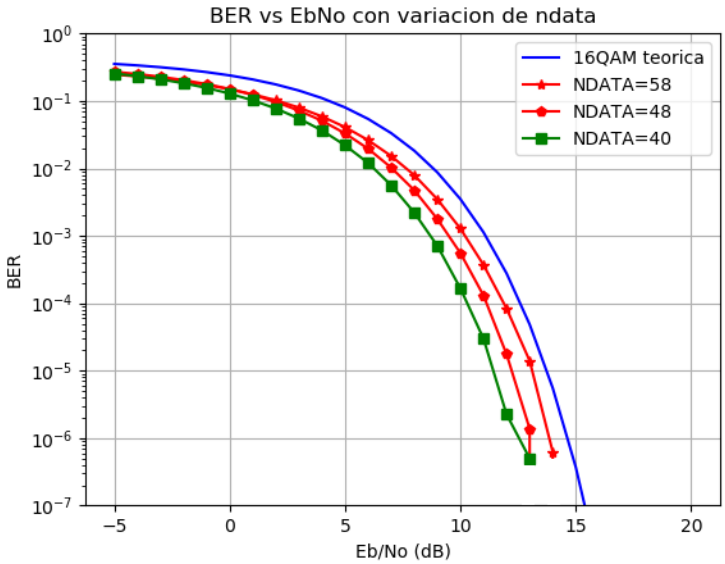


Figura 4-29. BER vs E_b/N_o para modulación OFDM con tamaño de la FFT fija y número de subportadoras de datos variable

Como observación, en el diseño de este sistema ,el número de subportadoras de datos será influyente en la BER ya que cuantas menos portadoras de datos hay menor será la probabilidad de error.

4.8.2. Esquema OFDM con prefijo cíclico

4.8.2.1. Nfft=64 y Natos=48

```
#%%Prueba 3. OFDM con nfft=64 y ndatos=48 para distintas modulaciones con CP
plt.figure(3)
plt.title('BER vs EbNo para modulacion OFDM con variacion de M y con CP')
BER_teorica_PSK(4,'b')
BER_teorica_QAM(16,'r')
BER_teorica_QAM(64,'g')
ofdm_BER(4,-5,20,'b',64,48,Ncp=10,modo=0)
ofdm_BER(16,-5,20,'r',64,48,Ncp=10,modo=1)
ofdm_BER(64,-5,20,'g',64,48,Ncp=10,modo=1)
plt.grid(True)
plt.legend(('QPSK teorica','16QAM teorica',
            '64QAM teorica','OFDM/QPSK',
            'OFDM/16QAM','OFDM/64QAM'),loc=1)
plt.xlabel('Eb/No (dB)')
plt.ylabel('BER')
plt.show()
```

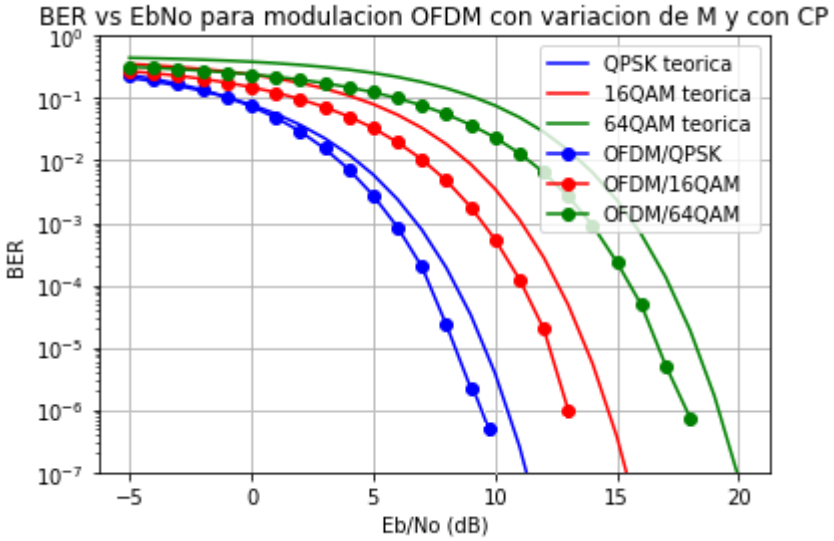


Figura 4-30. BER vs E_b/N_o para modulación OFDM con CP

Como se puede observar en la gráfica 4-30, no hay ninguna diferencia entre añadir o no prefijo cíclico

Incluir el prefijo cíclico afecta al espectro y a la tasa de transmisión, pero si el canal es AWGN sin multitrayecto no afecta a la BER. En el caso de multritrayecto la adición de prefijo cíclico mejorará la BER ya que favorecerá al sincronismo de la transmisión y, por ende, a que la calidad de la transmisión sea mejor que si no tuviese prefijo cíclico.

5 CONCLUSIONES

La implementación de un sistema de radiocomunicación mediante Python comparte la mayor parte del código para los distintos casos que se han visto con la excepción de la forma en la que se transmiten y reciben los datos, el número de muestras por símbolo y los tipos de modulación correspondientes.

Para poder desarrollar el sistema es necesario tener un buen conocimiento teórico sobre las comunicaciones digitales para poder detectar fallos y comprender qué se realiza en cada paso del sistema.

En el primer caso, implementando un sistema de radiocomunicación con filtrado raíz de coseno alzado para minimizar la ISI ya que este es de la familia del coseno alzado, se obtiene que los filtros están adaptados tanto en transmisión como en recepción. Con este caso se busca conseguir que la tasa de error de bit del sistema sea la mínima posible y se acerque lo más posible a la tasa de error de bit teórica.

Para ello se pueden implementar dos tipos de modulaciones, en este caso lineales, que son QAM y PSK. Las PSK son más eficientes para M más pequeñas además de ser más fáciles de implementar ya que la distancia entre todos los puntos es la misma. Sin embargo, para M más grandes conviene utilizar QAM ya que minimiza la probabilidad de error respecto a una M, puesto que al tener componentes en cuadratura y fase se pueden separar más los símbolos entre sí para energías más grandes.

Otro parámetro importante para este caso es la frecuencia utilizada para el filtro, ya que, a mayor frecuencia de muestreo, se obtendrá más precisión en las muestras transmitidas y, por ende, mejorará la probabilidad de error. Esta frecuencia debe cumplir las condiciones de Nyquist para minimizar la ISI introducida entre la transmisión de los símbolos. Con este parámetro se obtendrá el valor del número de muestras por símbolo añadidas.

El factor de *roll off* del filtro indica el ancho de banda que excede el filtro utilizado, por lo que a mayor factor de *roll off* se necesitará ocupar mayor ancho de banda para un resultado similar al de factores pequeños como se ha visto en las simulaciones de modo que si se quiere optimizar el sistema habrá que utilizar el menor factor de *roll off* posible en la medida que sea posible y el sistema lo permita.

Con respecto a la tasa de error por bit en este primer caso, se han conseguido los objetivos llegando a valores aproximados de la curva teórica. Teniendo en cuenta que se desea una probabilidad de error del orden de 10^{-6}, queda resumida en la siguiente tabla la relación energía por bit y ruido que se debe introducir.

Modulación	E_b/N_o (dB) Simulada	E_b/N_o (dB) Teórica
QPSK	10.2	10
8-PSK	13	13
8-QAM	13.5	12
16-PSK	18.5	18.5
16-QAM	15	14
32-PSK	-	23
32-QAM	18.5	19
64-PSK	-	28
64-QAM	18.5	-

Este proceso ha sido la base para poder seguir con el segundo caso, que consiste en implementar una modulación OFDM con QPSK o QAM. Los errores o dificultades de esta parte se han debido en gran parte a la integración de librerías por software libre. Se ha tenido que utilizar el filtro raíz de coseno

alzado y el canal AWGN de una librería externa (sk_dsp) a la que se iba a utilizar en un primer momento (scikit_commpy) ya que la raíz de coseno alzado estaba mal definida en aquella y el canal AWGN no daba la respuesta deseada. No obstante, no se ha tenido problema con los moduladores QAM y PSK, pudiendo acceder a los métodos de modular y demodular de forma sencilla. Es interesante observar las respuestas de las componentes en el tiempo para ver que todo vaya correctamente, de ahí se ha deducido la necesidad de normalizar la señal modulada recibida en la QAM ya que ese detector no es capaz de detectar los símbolos si no tienen la amplitud que deben tener como los símbolos transmitidos.

Tanto para OFDM/QAM como OFDM/PSK se mejora la BER simulada respecto a la BER teorica sin el empleo de OFDM por lo que, entre las dos implementaciones, esta es la mejor. Este esquema permite disminuir la complejidad hardware del sistema.

La ortogonalidad utilizando OFDM permite reducir el efecto de la distorsión producida por el multitrayecto ya que si se añade CP y este es mas largo que la respuesta impulsiva del canal se puede evitar los errores de ISI e ICI producidos por errores de sincronismo ya que pasa de tener una convolución lineal a una convolución circular. En este caso, no se ha considerado respuesta impulsiva del canal por lo que no se introduce el eco producido por el multitrayecto.

6 LINEAS FUTURAS

El sistema de radiocomunicación se ha presentado para unas características básicas como son el ruido gaussiano, añadiendo sobremuestreo y submuestreo o la simulación utilizando OFDM, pero se podría ampliar teniendo en cuenta los siguientes aspectos:

- Estudiar la interferencia por multritrayecto añadiendo respuestas impulsivas del canal: la distorsión lineal dada por la llegada de la señal por diferentes caminos introduce un desfase de tiempo entre ellas lo que empeorará la BER medida en el receptor. Para solucionarlo, se puede implementar un ecualizador adaptativo y uso de prefijo cíclico para el sincronismo.
- Añadir canal Rayleigh: al igual que el ruido Gaussiano, es un modelo estadístico presente sobre todo en las señales RF como el que se utiliza en comunicaciones inalámbricas.
- Implementación de corrección y detección de errores: para evitar la pérdida de paquetes, se desean códigos redundantes para que el receptor sea capaz de corregir los errores que se hayan originado en la transmisión. Por ejemplo, se podría utilizar el código de Hamming.
- Implementación de MIMO: aumento de la tasa de transferencia utilizando distintos canales en la transmisión de datos o multiplexación espacial de las antenas. Por ejemplo, en la tecnología 5G de comunicaciones móviles se pretende instalar MIMO masivo.
- Implementación de multiplexación: combinación de dos o más canales de información para el mismo medio de transmisión como FDM, TDM
- Implementación de variantes de OFDM como SC-FDMA o COFDM.
- Implementación de modulaciones no lineales.

Por simplicidad y con expectativas de futuro, es mejor programar una librería desde cero entendiéndola y asegurándose de que esté bien que coger librerías de código libre, ya que el código puede ser incorrecto y que las funciones no cumplan con las expectativas esperadas.

REFERENCIAS

[1] V. Taranalli, "CommPy: Digital Communication with Python", version 0.3.0. Available at https://github.com/veeresht/CommPy, 2015.

[2] M. Wickert , "Scikit-dsp-comm". Disponible en https://github.com/mwickert/scikit-dsp-comm, 2017,

[3] D. López "Desarrollo de herramientas para el control remoto de una fuente de alimentación mediante Python", Universidad de Sevilla, 2018.

[4] L. Jimenez, J.Parrado, C. Quiza, C.Suarez, "Modulación multiportadora OFDM", District University of Bogota y Universidad Distrital Franscisco José de Caldas, Vol 6, Nº1, 2001

[5] F. J. Payan Somet, "Principios de comunicaciones digitales. I, Fundamentos". Sevilla: Universidad de Sevilla, Secretariado de Publicaciones, 2014.

[6] B.Acha, "Apuntes Comunicaciones Digitales Avanzadas", Universidad de Sevilla

[7] K. M. Gharaibeh, "Nonlinear distortion in wireless systems : modelling and simulation with MATLAB". Hoboken, N.J: Wiley. 2012.

[8] Max. Basic OFDM Example in Python. In *DSP Illustrations*.

[9] I.Pulido "Control remoto de instrumentación para radiocomunicación mediante Python", Master's thesis, Universidad de Sevilla 2017.

[10] S. Haykin, "Sistemas de comunicación", México: Limusa Wiley, 2008

[11] M.C. Jeruchim, P.Balaban, K.S. Shanmugan, "Simulation of communication Second Edition: Systems.Modeling,Methology and Techniques" Kluwer Academic/Plenum Publishers, New York, 2000

[12] A.R. Castro Lechtaler, R.J. Fusario, "Comunicaciones y Redes. Para profesionales en sistemas de información". MARCOMBO, S.A 2015

[13] E. Bahit, "Curso: Python para principiantes", Disponible en http://librosweb.es/libro/python/ ,2012

[14] "Las Telecomunicaciones y la Movilidad en la Sociedad de la Información,Capítulo 3 ", 2014

[15] R Susthar, S. Joshi, N. Agrawal, "Performance Analysis of Different M-ARY Modulation Techniques in Cellular Mobile Communication"

GLOSARIO

DAB: Difusión de audio digital

DVB-T: Difusión de video digital terrestre

ISDB-T: Servicios Integrados digital terrestre

LAN: Red de Area Local

DRM: Radio digital de baja frecuencia

ADSL: Línea de Abonado Digital Asimétrica

IEEE: Institute of Electrical and Electronics Engineers

WiMax: Worldwide Interoperability for Microwave Access

OFDM: Modulacion de frecuencia ortogonal digital

SRC: Raiz de coseno alzado

WSS: Estacionario en Sentido Amplio

ANEXO 1. CÓDIGO

Implementación de un sistema de radiocomunicación con filtro raíz de coseno alzado y modulación QAM y PSK

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 25 13:38:52 2019
@author: Maria Teresa Jurado Cañero
"""

import numpy as np
from modulation import PSKModem #commpy.modulation
from modulation import QAMModem #commpy.modulation
from scipy.special import erfc
import matplotlib.pyplot as plt
from math import log,sin,pi,sqrt
from sigsys import downsample,upsample,cpx_AWGN #sk_dsp.scipy
from digitalcom import sqrt_rc_imp,rc_imp,Q_fctn #sk_dsp.digitalcom



def  PSK_BER(M,Ns,EbNo_MIN,EbNo_MAX,color,alpha):
    """
    Esta función devuelve las gráficas teóricas y simuladas de la BER
    en un sistema de radiocomunicaciones para una modulacion MPSK
    Dependiendo de los parámetros que se les pase evaluará el sistema
    de una forma distinta
    M: Número de símbolos de la constelación
    Ns: Número de bits transmitidos
    EbNo_MIN,EbNo_MAX: intervalos de evaluación de la BER
    color: parámetro añadido por estética para representar varias gráficas
    alpha: factor de roll-off de los filtros de transmisión y recepción
    """
    ##Definicion del intervalo de EbNo
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
```

```
##Se crea un objeto te tipo PSK
mod1=PSKModem(M)
k=mod1.num_bits_symbol
##Se crearán variables vacías para ir añadiendo los valores
Pe       = np.empty(np.shape(Eb_No_lin))
BER      = np.empty(np.shape(Eb_No_lin))

##Parámetros para el diseño del filtro
Ts=1;
Fs=8; #Cuanto más suba Fs, más se acercará la BER a la gráfica teórica
nsamp=Fs*Ts; #Muestras por simbolo

##Diseño del filtro
filtro= sqrt_rc_imp(nsamp,alpha)
##Crear el vector de datos binarios
x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
##Aplicar la modulacion PSK a los bits
m_tx=mod1.modulate(x_tx)
##Sobremuestrear y filtrar los datos de entrada
y_tx=upsample(m_tx,nsamp)
r_tx=np.convolve(y_tx,filtro,mode='same')
##Pasar la señal por un canal AWGN
loop=0;
for ebno in Eb_No_dB:
   error_sum=0;
   esno = ebno + 10*log(k,10)
   w=cpx_AWGN(r_tx,esno,nsamp)
   ##Filtrar y submuestrear
   y_rx=np.convolve(w,filtro,mode='same')
   ##Submuestrear la señal recibida
   m_rx= downsample(y_rx,nsamp)
  ##Demodular la señal recibida
   x_rx= mod1.demodulate(m_rx,'hard')
   total_sent=len(x_rx)
   ##Se comprueban los errores comparando la transmisión con la recepción
   for u,v in zip(x_tx,x_rx):
      if u!=v:
```

```
                error_sum+=1;
    ##Cálculo de la Tasa de Error por Bit
    BER[loop] = float(error_sum)/float(total_sent)
    loop += 1

loop=0;
##Cálculo de la probabilidad de error teórica
for en in Eb_No_lin:
    Pe[loop]  = (2/log(M,2))*Q_fctn(sqrt(2*log(M,2)*en)*sin(pi/M))
    loop+=1
plt.ylim(10**-7)
#plt.semilogy(Eb_No_dB, Pe,color,linewidth=2) Esta implementada en BER_teorica también.
plt.semilogy(Eb_No_dB, BER,color)
```

```
def QAM_BER(M,Ns,EbNo_MIN,EbNo_MAX,color,alpha):
    """
    Esta función devuelve las gráficas teóricas y simuladas de la BER
    en un sistema de radiocomunicaciones para una modulacion MQAM
    Dependiendo de los parámetros que se les pase evaluará el sistema
    de una forma distinta
    M: Número de símbolos de la constelación
    Ns: Número de bits transmitidos
    EbNo_MIN,EbNo_MAX: intervalos de evaluación de la BER
    color: parámetro añadido por estética para representar varias gráficas
    alpha: factor de roll-off de los filtros de transmisión y recepción
    """
    ##Definicion del intervalo de EbNo
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
    ##Se crea un objeto te tipo QAM
    mod1=QAMModem(M)
    k=mod1.num_bits_symbol
    ##Se crearán variables vacías para ir añadiendo informacion
    Pe       = np.empty(np.shape(Eb_No_lin))
```

```
BER      = np.empty(np.shape(Eb_No_lin))

##Parámetros para el diseño del filtro
Ts=1;
Fs=8; #Cuanto más suba Fs, más se acercará la BER a la gráfica teórica
nsamp=Fs*Ts; #Muestras por simbolo

##Diseño del filtro
filtro= sqrt_rc_imp(nsamp,alpha)
##Crear el vector de datos binarios
x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
##Aplicar la modulacion QAM a los bits
m_tx=mod1.modulate(x_tx)
##Sobremuestrear y filtrar los datos de entrada
y_tx=upsample(m_tx,nsamp)
r_tx=np.convolve(y_tx,filtro,mode='same')
##Pasar la señal por un canal AWGN
loop=0;
for ebno in Eb_No_dB:
   error_sum=0;
   esno = ebno + 10*log(k,10)
   w=cpx_AWGN(r_tx,esno,nsamp)
   ##Filtrar la señal tras pasarla por el canal
   y_rx=np.convolve(w,filtro,mode='same')
   ##Submuestrear la señal recibida y linealizarla
   m_rx= downsample(y_rx,nsamp)
   m_rx=m_rx/np.linalg.norm(m_rx)*np.linalg.norm(m_tx)
   ##Demodular la señal recibida
   x_rx= mod1.demodulate(m_rx,'hard')
   total_sent=len(x_rx)
   #Se comprueban los errores comparando la transmisión con la recepción
   for u,v in zip(x_tx,x_rx):
      if u!=v:
         error_sum+=1;
   ##Cálculo de la Tasa de Error por Bit
   BER[loop] = float(error_sum)/float(total_sent)
   loop+=1
```

```
##Cálculo de la probabilidad de error teórica
loop=0;
for en in Eb_No_lin:
    term1=(1-(1/sqrt(M)))*erfc(sqrt(3*en*k/(2*(M-1))))
    term2=(1-(0.5*(1-(1/sqrt(M))))*erfc(sqrt(3*en*k/(2*(M-1)))))
    Pe[loop] = term1*term2
    loop+=1
plt.ylim(10**-7)
#plt.semilogy(Eb_No_dB, Pe,color,linewidth=2)
plt.semilogy(Eb_No_dB, BER,color)
```

Implementación de un sistema de radiocomunicación con filtro raíz de coseno alzado y modulación OFDM

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 10 16:45:11 2019

@author: Teresa Jurado Cañero
"""

import matplotlib.pyplot as plt
from sigsys import cpx_AWGN
from scipy import signal
from numpy import array,arange
from math import log
import numpy as np
from ofdm_sk_2 import OFDM_rx,OFDM_tx
from scipy.special import erfc
from math import log,sin,pi,sqrt
from modulation import QAMModem,PSKModem
```

```
def ofdm_BER(M,EbNo_MIN,EbNo_MAX,color,nfft,nd,estado,Ncp=0,modo=1):
    """
    M indica el número de símbolos que hay en la constelación
    EbNo_MIN,EbNo_MAX son los intervalos de evaluación de la BER
    color: para la representación de distintas gráficas
    nfft: número total de subportadoras
    nd: número de portadoras con datos
    Ncp: longitud de prefijo cíclico
    modo=1 -->QAM(por defecto)
    modo=0 -->PSK(para 4PSK)
    """
    #Definición del intervalo de EbNo
    Ns=2000000
    Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
    Eb_No_lin = 10**(Eb_No_dB/10.0)
    #Se crearán variables vacias para ir añadiendo los resultados
    Pe= np.empty(np.shape(Eb_No_lin))
    BER= np.empty(np.shape(Eb_No_lin))
     ##Se crea un objeto te tipo QAM o PSK
    if modo==0:
        mod1=PSKModem(M)
    else:
        mod1=QAMModem(M)
    k=mod1.num_bits_symbol
     ##Crear el vector de datos binarios
    x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
    total_sent=len(x_tx) #Se utilizará para medir la BER
    ##Aplicar la modulacion QAM a los bits
    m_tx=mod1.modulate(x_tx)
    #Señal modulada mediante OFDM a transmitir
    ofdm_tx_symbols= OFDM_tx(m_tx,nd,nfft,0,estado,Ncp)
    ##Pasar la señal por un canal AWGN
    loop=0;
    for ebno in Eb_No_dB:
        error_sum=0;
        esno = ebno + 10*log(k,10)
```

```
    ofdm_rx_symbols=cpx_AWGN(ofdm_tx_symbols,esno,nfft//nd)

    ##Submuestrear la señal recibida y linealizarla
    m_rx= OFDM_rx(ofdm_rx_symbols,nd,nfft,0,estado,Ncp,0.95)
    ##Demodular la señal recibida
    x_rx= mod1.demodulate(m_rx,'hard')
    #Se comprueban los errores comparando la transmisión con la recepción
    for u,v in zip(x_tx,x_rx):
        if u!=v:
            error_sum+=1;
    ##Cálculo de la Tasa de Error por Bit
    BER[loop] = float(error_sum)/float(total_sent)
    loop += 1

##Cálculo de la probabilidad de error teórica
loop=0;
for en in Eb_No_lin:
    term1=(1-(1/sqrt(M)))*erfc(sqrt(3*en*k/(2*(M-1))))
    term2=(1-(0.5*(1-(1/sqrt(M)))))*erfc(sqrt(3*en*k/(2*(M-1)))))
    Pe[loop] = term1*term2
    loop+=1
plt.ylim(10**-7)
#plt.semilogy(Eb_No_dB, Pe,color,linewidth=2)
plt.semilogy(Eb_No_dB, BER,'-'+color)
```

ofdm_sk_dsp.py

```
from matplotlib import pylab
from matplotlib import mlab
import numpy as np
from numpy.fft import fft,ifft
import matplotlib.pyplot as plt
from scipy import signal
from scipy.special import erfc
from sys import exit
from sigsys import upsample
```

```
from sigsys import downsample
from sigsys import NRZ_bits
from sigsys import NRZ_bits2
from sigsys import PN_gen
from sigsys import m_seq
from sigsys import cpx_AWGN
from sigsys import CIC

def OFDM_tx(IQ_data, Nf, N, Np=0, cp=False, Ncp=0):
    Nf=int(Nf)
    N=int(N)
    Ncp=int(Ncp)
    """
    Parameters
    ----------
    IQ_data : +/-1, +/-3, etc complex QAM symbol sample inputs
    Nf : number of filled carriers, must be even and Nf < N
    N : total number of carriers; generally a power 2, e.g., 64, 1024, etc
    Np : Period of pilot code blocks; 0 <=> no pilots
    cp : False/True <=> bypass cp insertion entirely if False
    Ncp : the length of the cyclic prefix
    Returns
    -------
    x_out : complex baseband OFDM waveform output after P/S and CP insertion
    """
    N_symb = len(IQ_data)
    N_OFDM = N_symb // Nf
    IQ_data = IQ_data[:N_OFDM * Nf]
    IQ_s2p = np.reshape(IQ_data, (N_OFDM, Nf))  # Símbolos de subportadoras por columnas
    print(IQ_s2p.shape)
    if cp:
        x_out = np.zeros(N_OFDM * (N + Ncp), dtype=np.complex128)
    else:
        x_out = np.zeros(N_OFDM * N, dtype=np.complex128)
    for k in range(N_OFDM):
        buff = np.zeros(N, dtype=np.complex128)
        for n in range(-Nf // 2, Nf // 2 + 1):
```

```
            if n == 0:  # Subportadora modulada
                buff[0] = 0


            elif n > 0:  #  Subportadora modulada a f = 1:Nf/2
                buff[n] = IQ_s2p[k, n - 1]
            else:  # Subportadora modulada a f = -Nf/2:-1
                buff[N + n] = IQ_s2p[k, Nf + n]
        if cp: # Con prefijo ciclico
            x_out_buff = ifft(buff)
            x_out[k * (N + Ncp):(k + 1) * (N + Ncp)] = np.concatenate((x_out_buff[N - Ncp:],
                                                    x_out_buff))
        else:# Sin prefijo cíclico
            x_out[k * N:(k + 1) * N] = ifft(buff)
    return x_out




def OFDM_rx(x, Nf, N, Np=0, cp=False, Ncp=0, alpha=0.95, ht=None):
    Nf=int(Nf)
    N=int(N)
    Ncp=int(Ncp)
    """
    Parameters
    ----------
    x : Received complex baseband OFDM signal
    Nf : Number of filled carriers, must be even and Nf < N
    N : Total number of carriers; generally a power 2, e.g., 64, 1024, etc
    Np : Period of pilot code blocks; 0 <=> no pilots; -1 <=> use the ht impulse response input to equalize the
OFDM symbols; note equalization still requires Ncp > 0 to work on a delay spread channel.
    cp : False/True <=> if False assume no CP is present
    Ncp : The length of the cyclic prefix
    alpha : The filter forgetting factor in the channel estimator. Typically alpha is 0.9 to 0.99.
    nt : Input the known theoretical channel impulse response
    Returns
    -------
    z_out : Recovered complex baseband QAM symbols as a serial stream; as appropriate channel estimation
has been applied.
```

```
    N_symb = len(x) // (N + Ncp)


    y_out = np.zeros(N_symb * N, dtype=np.complex128)
    for k in range(N_symb):
        if cp:
            # Se quita el prefijo cíclico
            buff = x[k * (N + Ncp) + Ncp:(k + 1) * (N + Ncp)]
        else:
            buff = x[k * N:(k + 1) * N]
        y_out[k * N:(k + 1) * N] = fft(buff)
    #Demultiplexación en flujos de Nf paralelos de los N totales incluyendo las pilotos que contienen
información de canal
    z_out = np.reshape(y_out, (N_symb, N))
    z_out = np.hstack((z_out[:, 1:Nf // 2 + 1], z_out[:, N - Nf // 2:N]))
    return z_out.flatten()
```

Código para la representación del punto 4.2.

```
import numpy as np
from modulation import PSKModem
from modulation import QAMModem
from scipy.special import erfc
import matplotlib.pyplot as plt
from math import log,sin,pi,sqrt
from sigsys import downsample,upsample,cpx_AWGN
from digitalcom import sqrt_rc_imp,rc_imp,Q_fctn
#%%
EbNo_MIN=-5
EbNo_MAX=20
alpha=0.22
M=4
Ns=100000
##Definicion del intervalo de EbNo
Eb_No_dB  = np.arange(EbNo_MIN,EbNo_MAX+1)
Eb_No_lin = 10**(Eb_No_dB/10.0)
##Se crea un objeto te tipo PSK
mod1=PSKModem(M)
```

```
k=mod1.num_bits_symbol
##Se crearán variables vacías para ir añadiendo los valores
Pe      = np.empty(np.shape(Eb_No_lin))
BER     = np.empty(np.shape(Eb_No_lin))

##Parámetros para el diseño del filtro
Ts=1;
Fs=8; #Cuanto más suba Fs, más se acercará la BER a la gráfica teórica
nsamp=Fs*Ts; #Muestras por simbolo

##Diseño del filtro
filtro= sqrt_rc_imp(nsamp,alpha)
#%%Crear el vector de datos binarios
plt.figure(1)
plt.title('Ristra de bits')
plt.xlabel('t')
x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
t=np.arange(0,100)
plt.stem(t,x_tx[0:100])
#%%#Aplicar la modulacion PSK a los bits
m_tx=mod1.modulate(x_tx)
plt.figure(2)
plt.title('Ristra de bits modulada (Parte en cuadratura)')
plt.xlabel('t')
plt.stem(np.arange(0,100),m_tx[0:100].real)

#%%Sobremuestrear
y_tx=upsample(m_tx,nsamp)
plt.figure(3)
plt.xlabel('t')
plt.stem(np.arange(0,10),m_tx[0:10].real,'-b')
plt.title('Parte en cuadratura sobremuestreada y sin sobremuestrear')
plt.xlabel('t')
plt.stem(np.arange(0,10*nsamp),y_tx[0:10*nsamp],'-or')
plt.legend(('IQ','IQ sobremuestreados'))
#%%Filtrar los datos de entrada
```

```
r_tx=np.convolve(y_tx,filtro,mode='same')
plt.figure(4)
plt.title('Transmitision sobremuestreada')
plt.xlabel('t')
plt.plot(np.arange(0,100*nsamp),r_tx[0:100*nsamp])
#%%Pasar la señal por un canal AWGN PRUEBA 1
ebno=-5 #Con mucho ruido
#ebno=20 #Con poco ruido --> PROBARLO LOS DOS
error_sum=0;
esno = ebno + 10*log(k,10)
w1=cpx_AWGN(r_tx,esno,nsamp)
plt.figure(5)
plt.xlabel('t')
plt.plot(np.arange(0,100*nsamp),w1[0:100*nsamp])
y_rx1=np.convolve(w1,filtro,mode='same')
##Submuestrear la señal recibida
m_rx1= downsample(y_rx1,nsamp)
##Demodular la señal recibida

x_rx1= mod1.demodulate(m_rx1,'hard')

#%%Pasar la señal por un canal AWGN PRUEBA 2
ebno=20 #Con poco ruido --> PROBARLO LOS DOS
error_sum=0;
esno = ebno + 10*log(k,10)
w2=cpx_AWGN(r_tx,esno,nsamp)
plt.figure(5)
plt.xlabel('t')
plt.plot(np.arange(0,100*nsamp),w2[0:100*nsamp])
y_rx2=np.convolve(w2,filtro,mode='same')
##Submuestrear la señal recibida
m_rx2= downsample(y_rx2,nsamp)
##Demodular la señal recibida
plt.figure(6)
x_rx2= mod1.demodulate(m_rx2,'hard')
plt.stem(t[0:50],x_rx2[0:50])
```

```
#%%
plt.figure(7)
plt.xlabel('t')
plt.plot(np.arange(0,100*nsamp),y_rx2[0:100*nsamp]/np.linalg.norm(y_rx2)*np.linalg.norm(r_tx))
plt.plot(np.arange(0,100*nsamp),r_tx[0:100*nsamp],'--')
#%%
plt.figure(8)
plt.xlabel('t')
plt.title('Parte en cuadratura sobremuestreada y sin sobremuestrear')
plt.xlabel('t')
plt.stem(np.arange(0,5*nsamp),y_rx2[0:5*nsamp],'-r')
plt.stem(np.arange(1,5),m_rx2[1:5].real,'-b')
plt.stem(np.arange(0,5*nsamp)[0:5*nsamp:nsamp],np.zeros(5),'o')
plt.legend(('IQ','IQ submuestreados'))
#%%
plt.figure(9)
plt.xlabel('t')
error=np.zeros(len(x_tx))
bandera=0;
for u,v in zip(x_tx,x_rx1):
    if u!=v:
        error[bandera]=1.
    else:
        error[bandera]=0.
    bandera+=1
plt.stem(t[0:50],error[0:50])
```

Código para la representación del punto 4.3

```
import matplotlib.pyplot as plt
from sigsys import cpx_AWGN
from scipy import signal
from numpy import array,arange
from math import log
import numpy as np
from ofdm_sk_2 import OFDM_rx,OFDM_tx
from scipy.special import erfc
```

```
from math import log,sin,pi,sqrt
from modulation import QAMModem,PSKModem

EbNo_MIN=-5
EbNo_MAX=20
nd=48
nfft=64
cp=False
Ncp=10
M=16
#Definición del intervalo de EbNo
Ns=100
Eb_No_dB = np.arange(EbNo_MIN, EbNo_MAX+1)
Eb_No_lin = 10**(Eb_No_dB/10.0)
#Se crearán variables vacías para ir añadiendo los resultados
Pe= np.empty(np.shape(Eb_No_lin))
BER= np.empty(np.shape(Eb_No_lin))
 ##Se crea un objeto te tipo QAM
mod1=QAMModem(M)
k=mod1.num_bits_symbol
 ##Crear el vector de datos binarios
x_tx = np.random.randint(0, 2, k*Ns) # Ristra de bits binaria y aleatoria
total_sent=len(x_tx) #Se utilizará para medir la BER
##Aplicar la modulación QAM a los bits
m_tx=mod1.modulate(x_tx)
#%%Señal modulada mediante OFDM a transmitir
ofdm_tx_symbols= OFDM_tx(m_tx,nd,nfft,0,cp,2)

plt.psd(ofdm_tx_symbols,2**10,1);
plt.xlabel('Frecuencia normalizada($\omega/(2\pi)=f/f_s$)')
plt.ylim([-40,0])
plt.xlim([-.5,.5])
plt.show()
#%%Pasar la señal por un canal AWGN
loop=0;
ebno=20 # ebno=0 o ebno=10
esno = ebno + 10*log(k,10)
```

```
ofdm_rx_symbols=cpx_AWGN(ofdm_tx_symbols,esno,nfft//nd)
plt.psd(ofdm_rx_symbols,2**10,1);
plt.xlabel('Normalized Frequency ($\omega/(2\pi)=f/f_s$)')
plt.ylim([-40,0])
plt.xlim([-.5,.5])
plt.show()
##Submuestrear la señal recibida y linealizarla
m_rx= OFDM_rx(ofdm_rx_symbols,nd,nfft,0.cp,Ncp,0.95)
plt.plot(m_rx.real,m_rx.imag,'o') #Hacerlo para distintos valores de EbNo
##Demodular la señal recibida
x_rx= mod1.demodulate(m_rx,'hard')
#S#e comprueban los errores comparando la transmisión con la recepción
for u,v in zip(x_tx,x_rx):
    if u!=v:
        error_sum+=1;
##Cálculo de la Tasa de Error por Bit
BER[loop] = float(error_sum)/float(total_sent)
loop += 1
```