27 DE DICIEMBRE DE 2024

PROYECTO PRIME

SISTEMAS DE COMUNICACIÓN I

SERGIO RODRÍGUEZ GARCÍA, MIGUEL TORRES VALLS
LABORATORIO DE COMUNICACIONES
Profesor: Julio García Romero

Tabla de contenido

Introducción

El proyecto PRIME consiste en la implementación end-to-end de un sistema de comunicación basado en el estándar PoweRline Intelligent Metering Evolution, ITU-T G.9904, también conocido como estándar PRIME. Para el desarrollo de esta práctica, no se ha tenido en cuenta la parte del datagrama destinado al preámbulo, cabecera y CRC.

Se ha dividido la implementación del sistema en los siguientes apartados. En cada uno de ellos se explicarán los objetivos, desarrollo, resultados y conclusiones.

El sistema de transmisión está compuesto por:

- Codificador convolucional: convierte una serie de bits a la entrada de tamaño k en una palabra de bits a la salida de tamaño n, con n > k. El sistema tiene memoria por lo que una salida depende de las entradas anteriores.
- Aleatorización: operación lógica XOR entre los bits a transmitir y un vector de aleatorización conocido tanto por el transmisor como receptor. El objetivo es conseguir una frecuencia de cambio alta para que la señal pueda pasar los filtros paso alto del circuito transmisor.
- Entrelazado: se utiliza para reducir el efecto de errores burst. Estos errores se dan cuando el canal tiene desvanecimientos causados por fast o slow fading. Consiste en modificar el orden de transmisión para que, en caso de desvanecimiento del canal, afecte al menor número posible de bits de una misma trama.
- Modulación subportadora: modulación de cada una de las subportadoras de OFDM. Para ello, realizaremos todos los casos que se incluyen en el estándar Prime: dbpsk, dqpsk y d8psk.
- Modulación OFDM: modulación de las subportadoras en símbolos OFDM, donde las subportadoras sean ortogonales entre sí. Esto significa que los picos de cada subportadora deben estar alineados con los ceros de sus vecinas.

Por el contrario, el sistema de recepción lo forman:

- Ecualización: utiliza mensajes conocidos por el receptor, llamados pilotos para estimar el canal y contrarrestar sus efectos.
- Demodulación OFDM: obtiene los símbolos OFDM transmitidos.
- Demodulación subportadoras: a partir de los símbolos OFDM, demodula cada una de las subportadoras.
- Desentrelazado: realiza la operación inversa del entrelazado para reordenar las tramas recibidas.
- Desaleatorización: revierte los efectos del bloque aleatorización empleado en el transmisor. Para ello se realizará la operación XOR con el mismo vector de aleatorización que se utilizó en el transmisor.
- Detección y corrección de errores: gracias a la codificación de los bits a transmitir, en decodificación es posible detectar y corregir un número máximo de bits.

Implementación de todos los modos de comunicación de PRIME en el caso de canal sin distorsión y sin FEC

Objetivo

En este apartado se estudia la transmisión en un canal ideal (no dispersivo) sin prefijo cíclico ni corrección de errores (FEC) para los modos DBPSK, DQPSK y D8PSK definidos en el estándar. Se seleccionan y justifican los parámetros de simulación (número de bits para lograr BER fiable, número de símbolos OFDM por trama y número de tramas "payload") y se comparan los resultados de BER frente a SNR con los valores teóricos en un canal AWGN. Además, se comprueba la correcta recepción (ausencia de errores) tanto en ausencia de ruido como al introducir los bloques de aleatorización/desaleatorización, verificando así el comportamiento esperado de la cadena de transmisión sin FEC.

Parámetros de simulación

Para asegurar que la estimación de la BER resulte estadísticamente significativa hasta valores de 10^{-4} o incluso inferiores, se ha elegido transmitir un volumen de bits lo bastante grande en cada modo de modulación. En particular, se ha fijado un total de 10.000 bits como mínimo por modo (DBPSK, DQPSK, D8PSK) de forma que, al realizar la comparación de curvas BER vs. SNR, se disponga de suficientes errores (o ausencia de ellos) para lograr estadísticas fiables. En la práctica, veremos que este número será mayor, con el fin de garantizar la transmisión de más de una trama por modulación.

El estándar admite hasta 63 símbolos OFDM de información por trama y, además, se estima que el canal puede considerarse estable durante aproximadamente $100m_s$. Según las especificaciones PRIME, el periodo de un símbolo de información o payload es de $2{,}24m_s$, por lo que, para cumplir con la estabilidad del canal, utilizaremos 44 símbolos. Esto resulta en una duración total de $44 * 2{,}24 = 98{,}56m_s$

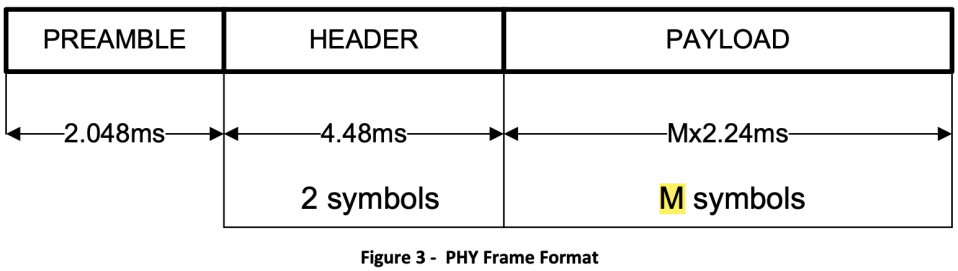

Figure 3 - PHY Frame Format

Ilustración 1 Periodo símbolo datagrama

De esta manera, se aprovecha al máximo el límite de estabilidad de canal, asegurando al mismo tiempo un tamaño suficiente de datos en cada trama para realizar un procesamiento estadísticamente representativo.

A fin de no depender de un único bloque o trama, se transmiten varias tramas de información para cada esquema de modulación. Para ello, se han fijado dos tramas para D8PSK, resultando en un total de 25344 bits. Para DQPSK y DBPSK se ha elegido el número de tramas correspondiente para alcanzar los mismos bits totales. Con ello, cada modo transmite el mismo volumen de bits, lo que permite comparar su rendimiento de manera equitativa y, al mismo tiempo, garantiza que se simule más de una trama para obtener estadísticas más robustas.

- Número de tramas D8PSK: 2
- Número de tramas DQPSK: 3
- Número de tramas DBPSK: 6

El número de bits de información por cada símbolo OFDM se ha obtenido siguiendo la tabla de a continuación, con el modo de código convolucional como "OFF". Los resultados han sido los siguientes:

- Bits D8PSK: $288 * 44 = 12672$
- Bits DQPSK: $192 * 44 = 8448$
- Bits DBPSK: $96 * 44 = 4224$

	DBPSK		DQPSK		D8PSK	
Convolutional Code (1/2)	On	Off	On	Off	On	Off
Information bits per subcarrier N_{BPSC}	0.5	1	1	2	1.5	3
Information bits per OFDM symbol N_{BPS}	48	96	96	192	144	288
Raw data rate (kbps approx)	21.4	42.9	42.9	85.7	64.3	128.6
Maximum MSDU length with 63 symbols (in bits)	3016	6048	6040	12096	9064	18144
Maximum MSDU length with 63 symbols (in bytes)"	377	756	755	1512	1133	2268

Ilustración 2 Tabla bits de información por modulaciones

El código utilizado para definir las longitudes de cada trama y el número de tramas necesarias para cada modulación es el siguiente:

```
%d8psk
N_d8psk = 288*M;
% Fijamos número de tramas d8psk a 2
N_tramas_d8psk = 2;
N_bits_totales = N_d8psk*N_tramas_d8psk;

%dqpsk
N_dqpsk = 192*M;
N_tramas_dqpsk = ceil(N_bits_totales/N_dqpsk);

% dbpsk
N_dbpsk = 96*M;
N_tramas_dbpsk = ceil(N_bits_totales/N_dbpsk);
```

Ausencia de errores sin ruido

En condiciones ideales, sin ruido (SNR muy alta) y sin prefijo cíclico, la transmisión y recepción de datos empleando solo los bloques de modulación diferencial (DPSK) junto con las transformadas IFFT y FFT no introduce errores porque la información se codifica en la diferencia de fase entre símbolos sucesivos. Concretamente, al modulador DPSK se le fija una fase de referencia para el primer símbolo (por ejemplo, 0 radianes), y cada símbolo posterior se transmite ajustando su fase en función de la anterior; de este modo, el bit o bits (según la modulación DBPSK, DQPSK, D8PSK, etc.) se representan por la variación de fase entre muestras consecutivas. En el receptor, el demodulador DPSK recupera los bits midiendo nuevamente la diferencia de fase entre cada par de símbolos recibidos, sin precisar un sincronismo de fase global como en las modulaciones coherentes. Por tanto, siempre que el canal sea ideal (h[n] = [1]) y no exista ruido, la fase diferencial recibida coincide exactamente con la transmitida, resultando en BER = 0. Este sistema de modulación tiene el inconveniente que un error en la detección de fase se acarrea en símbolos sucesivos.

Comprobamos que, como era de esperar, con ausencia de ruido no hay errores en la transmisión.

```
>> errores = sum(abs(bits_recibidos_concatenados-bits_transmitidos_concatenados))

errores =

     0
```

Ilustración 3 Errores en caso de canal ideal

Ausencia de errores sin ruido incluyendo aleatorización

Al añadir los bloques de aleatorización y desaleatorización a la cadena anterior, se preserva igualmente la ausencia de errores siempre que no haya ruido. La aleatorización simplemente redistribuye los bits de entrada según una secuencia pseudoaleatoria conocida en el receptor; así, la desaleatorización revertirá este proceso exactamente cuando la secuencia de aleatorización sea la misma, obteniendo la secuencia original de bits sin pérdidas. En el código MATLAB, esto se observa en las funciones aleatorizacion y en el uso de su vector inverso en demod_todo. Mientras la SNR sea teóricamente infinita (o muy alta en la práctica) y el canal permanezca libre de distorsiones, la traza de bits recibida coincide con la transmitida, corroborando la efectividad de la aleatorización/desaleatorización sin introducir errores adicionales.

El código Matlab que define la función aleatorización es el siguiente, donde la variable bits_aleatorio es el vector de aleatorización base proporcionado en la configuración PRIME. A partir del mismo, se construye el vector de aleatorización definitivo, concatenando muestras sucesivas.

```
function [bits_out, vector_aleatorizacion_out] =
aleatorizacion(bits_in)
bits_aleatorios = [0 0 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 0 1 0 0
0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1
```

```
1 0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0
1 1 1 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 1 1];
repeticiones_bits_aleatorios =
length(bits_in)/length(bits_aleatorios);
vector_aleatorizacion =
repmat(bits_aleatorios,1,ceil(repeticiones_bits_aleatorios))';

vector_aleatorizacion_out = vector_aleatorizacion(1:length(bits_in));
bits_out = xor(bits_in, vector_aleatorizacion_out);
end
```

El objetivo de la aleatorización es conseguir una frecuencia de cambio alta para que la señal pueda pasar los filtros paso alto del circuito transmisor y receptor. De lo contrario, una secuencia de muchos ceros (por ejemplo, cuando nos quedamos callados en mitad de una llamada), podría suponer en una señal con frecuencia muy baja incapaz de pasar los filtros de desacoplo entre etapas del circuito transmisor.

Una vez más, comprobamos la ausencia de errores con la fórmula anterior.

```
>> errores = sum(abs(bits_recibidos_concatenados-bits_transmitidos_concatenados))

errores =

     0
```

Ilustración 4 Errores en caso canal ideal con aleatorización

Curvas BER frente SNR

Para un canal AWGN invariante en frecuencia, las curvas BER vs SNR simuladas en el código (almacenadas en los vectores BER para DBPSK, DQPSK y D8PSK) muestran una tendencia muy próxima a las curvas teóricas (almacenadas en BER_teor). Conforme se incrementa la SNR, la BER simulada se reduce como era de esperar. Sin embargo, existen variaciones en la rapidez con la que se reduce la BER en función de la modulación.

En el caso de DBPSK, observamos que el último valor no nulo de BER se da con una relación señal a ruido de $9dB$, mientras que para DQPSK, se da con $13dB$ y para D8PSK entre $19dB$ y $21dB$.

Esto tiene sentido, ya que el ruido afecta en mayor medida a aquellas constelaciones que tienen menor distancia entre símbolos. Es mucho más probable cometer un error en detección si la distancia entre símbolos es 45º (caso D8PSK) que si es 180º (caso DBPSK). Es por esto, por lo que se consigue mucha mejor BER con menor SNR en constelaciones con menor número de bits por símbolo.

El código Matlab utilizado para calcular el BER asociado a cada SNR es el siguiente. Este código está formado por 3 bucles for anidados, el primero de ellos itera para todas las SNRs, el segundo para todas las modulaciones y el tercero para todas las tramas correspondiente a cada modulación.

```
% Vectores auxiliares para bucle for
N_tramas = [N_tramas_dbpsk, N_tramas_dqpsk, N_tramas_d8psk];
L_tramas = [N_dbpsk, N_dqpsk, N_d8psk];

Nfft = 512; % Número de puntos fft. Número de subportadores
disponibles
Nofdm = M; % Número de símbolos ofdm
Nf = 96; % Número de subportadoras

% Vector de SNRs
SNR_dB = -5:2:40;
offset_SNR = 10*log10(Nfft/(2*Nf));

% Vectores de BER calculada y BER teórica
BER = zeros(3, length(SNR_dB));
BER_teor = zeros(3, length(SNR_dB));

% Bucle que itera para todas las SNR
for snr=1:length(SNR_dB)
    % Bucle que itera para todas las modulaciones
    for i=1:3
        l_trama = L_tramas(i);
        n_tramas = N_tramas(i);

        bits_recibidos = zeros(n_tramas,l_trama);
        bits_transmitidos = zeros(n_tramas,l_trama);
        % Bucle que itera para todas las tramas
        for j=1:n_tramas
            % Si modulación de subportadora es DBPSK
            if i == 1
                tx_bits = randi([0,1],l_trama,1);

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_bits);

                [x_ofdm, piloto] = mod_todo(2, tx_aleatorio, Nfft,
Nofdm, Nf,0,0);

                x_ruido = awgn(x_ofdm, SNR_dB(snr)-
offset_SNR,'measured');
                rx_bits = demod_todo(2, x_ruido, Nfft, Nofdm, Nf,
vector_aleatorizacion,0,0);

            % Si la modulación de subportadora es DQPSK
            elseif i == 2
                tx_bits = randi([0,1],l_trama,1);

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_bits);

                [x_ofdm, piloto] = mod_todo(4, tx_aleatorio, Nfft,
Nofdm, Nf,0,0);

                x_ruido = awgn(x_ofdm, SNR_dB(snr)-
offset_SNR,'measured');
                rx_bits = demod_todo(4, x_ruido, Nfft, Nofdm, Nf,
vector_aleatorizacion,0,0);
```

```
            % Si la modulación de subportadora es D8PSK
            elseif i == 3
                tx_bits = randi([0,1],l_trama,1);

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_bits);

                [x_ofdm, piloto] = mod_todo(8, tx_aleatorio, Nfft,
Nofdm, Nf,0,0);

                x_ruido = awgn(x_ofdm, SNR_dB(snr)-
offset_SNR,'measured');
                rx_bits = demod_todo(8, x_ruido, Nfft, Nofdm, Nf,
vector_aleatorizacion,0,0);
            end
            % Guardo tramas recibidas en una matriz
            bits_recibidos(j,:) = rx_bits;
            bits_transmitidos(j,:) = tx_bits;
        end
        % Concateno filas de la matriz
        bits_recibidos_concatenados =
reshape(bits_recibidos,1,N_bits_totales);
        bits_transmitidos_concatenados =
reshape(bits_transmitidos,1,N_bits_totales);

        % Cálculo BER para todas las tramas de la modulación
        BER(i,snr) = sum(abs(bits_recibidos_concatenados-
bits_transmitidos_concatenados))/N_bits_totales;

        % Cálculo BER teórica para todas las tramas de la modulación
        if i == 1
            BER_t = DBPSK_BER(SNR_dB(snr));
        elseif i == 2
            BER_t = DQPSK_BER(SNR_dB(snr));
        elseif i == 3
            BER_t = D8PSK_BER(SNR_dB(snr));
        end
        BER_t(BER_t<1e-5)=NaN;
        BER_teor(i,snr) = BER_t;
    end
end
```

A continuación, se muestran las curvas completas de BER vs SNR teórica y simulada para cada modulación.

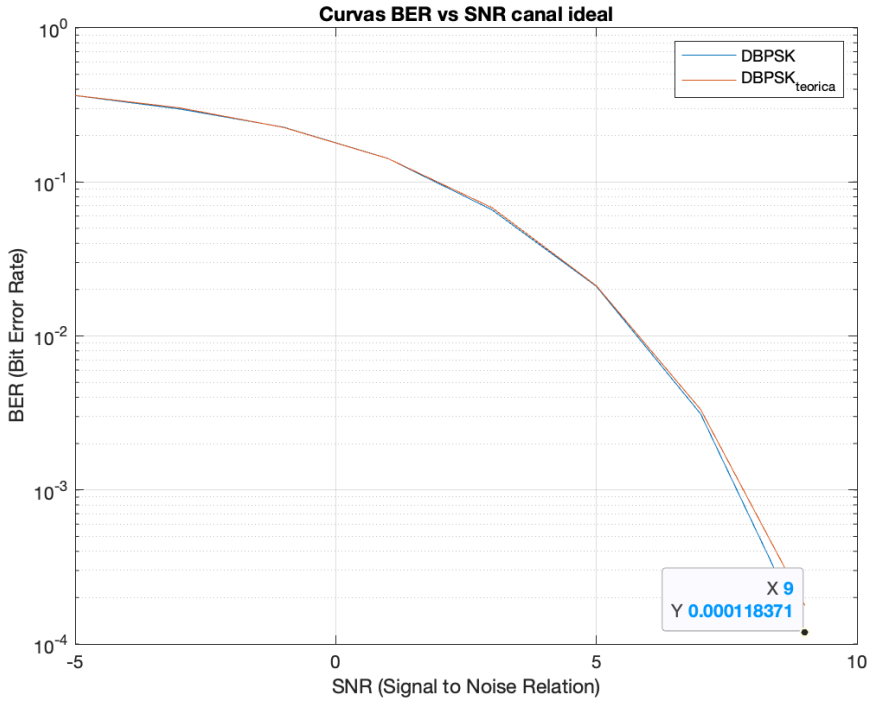

Ilustración 5 BER vs SNR canal ideal DBPSK

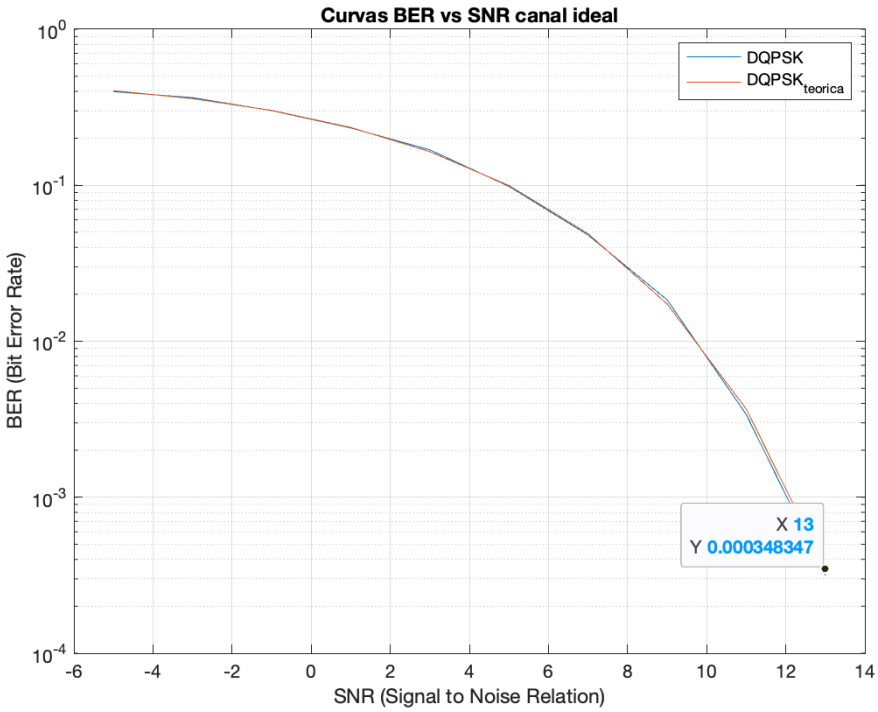

Ilustración 6 BER vs SNR canal ideal DQPSK

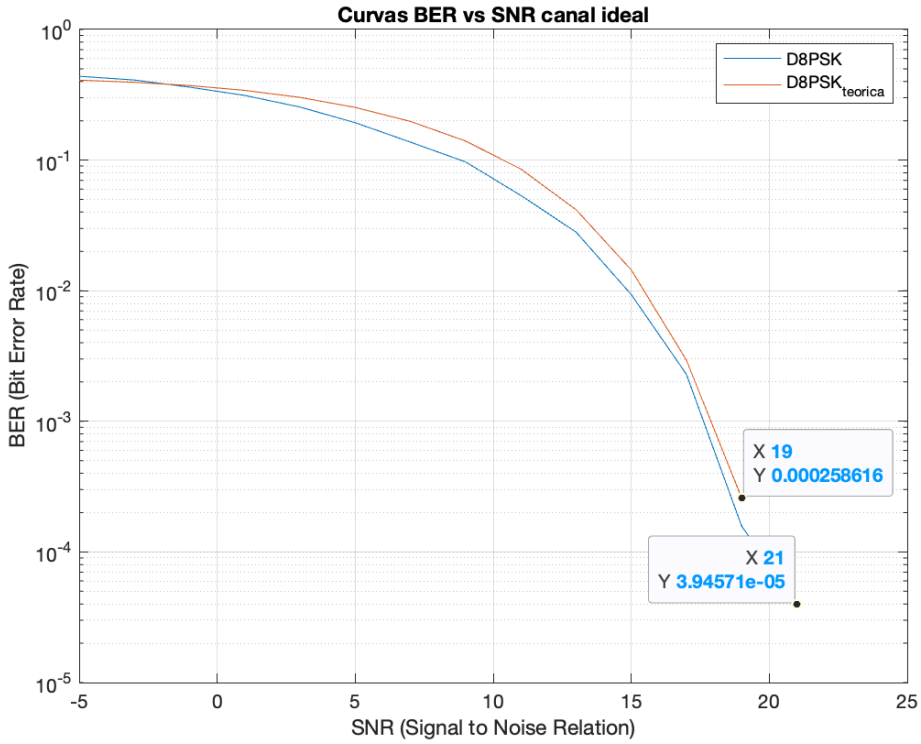

Ilustración 7 BER vs SNR canal ideal D8PSK

Conclusiones

En conclusión, los resultados obtenidos confirman el comportamiento esperado para un sistema OFDM con modulaciones diferenciales (DBPSK, DQPSK y D8PSK) en un canal ideal no dispersivo y sin FEC: (1) se observa ausencia de errores en condiciones libres de ruido, tanto sin aleatorización como incluyendo los bloques de aleatorización/desaleatorización, lo que demuestra la correcta implementación de la modulación diferencial y la aleatorización de la secuencia de bits; (2) la comparación de las curvas BER simuladas frente a las teóricas para un canal AWGN evidencia que la implementación reproduce adecuadamente la degradación de la BER según la SNR, con un acercamiento progresivo a la BER nula conforme se incrementa la relación señal a ruido; y (3) tal como era previsible y hemos visto en teoría, DBPSK presenta una mayor robustez en SNR bajas respecto a DQPSK y D8PSK, dado que la distancia entre símbolos en fase es mayor, mientras que D8PSK obtiene velocidades de transmisión más elevadas a costa de requerir mayores SNR para mantener niveles de error aceptables. De este modo, se valida la pertinencia de estos modos de modulación sin FEC en escenarios de canal ideal y se sientan las bases para su posterior estudio en entornos más realistas con dispersión y/o fading.

Inclusión de modelo de canal e inclusión de ecualizador

Objetivo

El objetivo principal de este apartado es estudiar el efecto de la dispersión del canal (mediante la respuesta al impulso $h[n]$) en una transmisión OFDM y comprobar cómo mejoran las prestaciones del sistema al incluir técnicas de mitigación. Concretamente, se requiere: (1) describir la señal recibida a partir de la convolución con el canal; (2) analizar la respuesta en frecuencia del canal y representarla en dB; (3) implementar el canal en el código y observar el deterioro de la BER cuando no se utilizan ni prefijo cíclico ni ecualización; y, finalmente, (4) incluir prefijo cíclico y un ecualizador (estimado con un símbolo piloto) para verificar la reducción de errores y su aproximación a la referencia ideal del canal AWGN.

Expresión algebraica de la señal recibida

Suponiendo que la señal transmitida es $s(t)$ y el canal tiene una respuesta al impulso discreta $h[n]$ la señal recibida puede describirse, en el dominio continuo, por la convolución de $s(t)$ con el canal: $r(t) = \int_{-\infty}^{\infty} \mathrm{s}(\tau) * \mathrm{h}(\mathrm{t}-\tau)\,\mathrm{d}\tau$.

En el dominio discreto, la convolución sigue la fórmula: $y[n] = \sum_{k=0}^{N-1} h[k] \cdot s[n-k]$, donde N es la longitud total del canal y $s[n]$ la versión muestreada de la señal a transmitir.

Función de transferencia del canal

El canal tiene una longitud total de 9 muestras no nulas, representado por el siguiente vector:

n	n<0	0	1	2	3	4	5	6	7	8	n>8
$h[n]$	0	-0.1	0.3	-0.5	0.7	-0.9	0.7	-0.5	0.3	-0.1	0

Ilustración 8 Muestras canal

Para su representación se han seguido los siguientes pasos:

- Realización de la DFT del canal utilizando el algoritmo FFT con un número total de puntos = 512
- Convertir la magnitud de la FFT en dB
- Vector de frecuencias. Para ello, es necesario: (1) crea un vector de la longitud total del canal (512) de 0 a 512, (2) dividirlo por 512 para normalizarlo, (3) restar 0,5 para centrarlo en el 0 y (4) multiplicar por la frecuencia de muestreo.

La frecuencia de muestreo se obtiene a partir de la duración del símbolo (T_{OFDM})y el número de puntos ($NFFT$). Estos valores los podemos obtener de las especificaciones PRIME a partir de la siguiente tabla.

FFT interval (samples	512
FFT interval (µs)	2048
Cyclic Prefix (samples)	48
Cyclic Prefix (µs)	192
Symbol interval (samples)	560
Symbol interval (µs)	2240

Ilustración 9 Tabla información características PRIME

Por último, a través de la siguiente ecuación obtenemos la frecuencia de muestreo.

$$T_{OFDM} = NFFT \cdot T_s \rightarrow T_s = \frac{T_{OFDM}}{NFFT} = \frac{2240\mu_s}{512} \rightarrow f_s = \frac{1}{T_s}$$

El código Matlab para la representación del canal es el siguiente:

```
Nfft = 512; % Número de puntos para la FFT
H = fft(h, Nfft); % Transformada en frecuencia del canal
H_dB = 20 * log10(abs(H)); % Magnitud en dB
fs = 250e3; % A partir de número de muestras FFT y periodo de símbolo OFDM
f = ((0:length(H)-1)/length(H)-0.5)*fs;
```

La representación del canal en el dominio de la frecuencia se muestra a continuación. Observamos el efecto del fading, amplificando unas frecuencias más que otras.

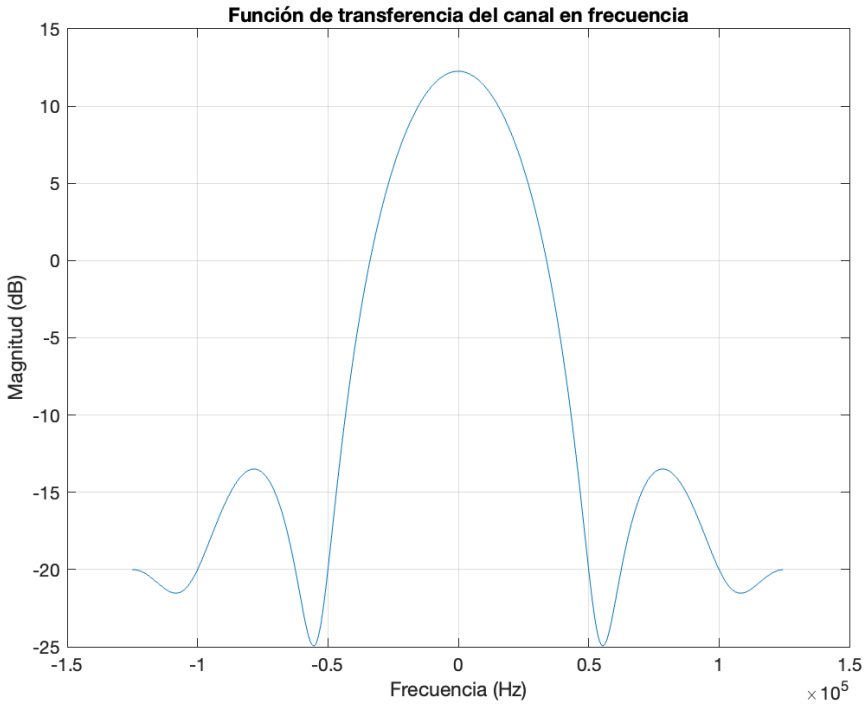

Ilustración 10 Representación en frecuencia del canal

Curvas BER vs SNR sin prefijo cíclico ni ecualización

Al modificar el código anterior para incluir la convolución con $h[n]$ y, posteriormente, aplicar la modulación OFDM convencional (sin prefijo cíclico ni ecualización), se obtienen unas curvas de BER vs SNR que empeoran en comparación con el caso ideal de canal plano. Esto ocurre porque el canal introduce una distorsión en frecuencia que el receptor no compensa, generando una interferencia significativa en las subportadoras.

El resultado es un BER más grande que el teórico, degradándose según aumenta la SNR hasta un cierto punto, y a partir de ahí se mantiene constante. Esto puede deberse a que en una modulación DPSK, un error en fase de un símbolo se acarrea en el siguiente, manteniendo sucesivos símbolos erróneos.

Al igual que en el bloque anterior, volvemos a notar diferencias significativas en función de la modulación utilizada. Para el caso de DBPSK, observamos que el último valor no nulo de la BER en el caso teórico se da con una SNR de $9\,dB$. Con esa misma SNR tenemos una BER simulada mucho mayor, aproximadamente de $0{,}11$. Esto se explica mediante el efecto del canal en nuestro sistema. Por último, observamos que la SNR a partir de la cual la BER simulada se mantiene estable es $21dB$, correspondiente con una $BER = 0{,}007$.

Si comparamos estos resultados con el resto de las modulaciones, observamos que para el caso de DQPSK, el último valor de BER teórico no nulo es el correspondiente a $13dB$. Con esa misma SNR el sistema presenta una BER simulada de $0{,}09$, lo que supone una variación de $0{,}02$ respecto a DBPSK. Sin embargo, el valor de SNR a partir el cual el BER se mantiene constante es de $25dB$, $4dB$ mayor que en el caso de DBPSK. Además, el valor de BER para $SNR \geq 25dB$, es de $0{,}01$, mayor que en el caso de DBPSK.

Estas variaciones se acentúan en mayor medida para el caso de D8PSK, observamos en su gráfica que el último valor de BER teórica no nulo se da con una SNR de $19dB$, correspondiente a una BER simulada de $0{,}07$. Esto supone una variación de $0{,}2$ respecto a DQPSK y $0{,}4$ respecto a DBPSK. Para el caso de la estabilidad de la BER, esta se da con $SNR \geq 29dB$, $4dB$ mayor que DQPSK y $8dB$ mayor que DBPSK. La BER simulada con esta SNR es de $0{,}025$, la más alta de las tres.

Podemos comprobar que hay cierto patrón que se cumple entre modulaciones en las dos métricas comparadas. El valor de SNR con el que se obtiene la última BER teórica no nula presenta una variación de $0{,}2$ en su valor de BER simulada. DQPSK disminuye $0{,}2$ respecto a DBPSK y D8PSK disminuye $0{,}2$ respecto a DQPSK, siendo menor en el caso de modulación D8PSK. Por el contrario, el valor de SNR para el cual conseguimos una BER estable es el menor en DBPSK, siendo aproximadamente de $21dB$. Este valor también se ve aumentado por un número constante en cada una de las modulaciones, siendo $25dB$ en DQPSK y $29dB$ en D8PSK.

El código Matlab para este apartado incluye el efecto del canal, añadiendo el ruido correspondiente a cada SNR. A continuación, se muestra el código utilizado para la

modulación DBPSK. Para el resto de las subportadoras, únicamente se ha cambiado el primer parámetro de las funciones mod_todo y demod_todo a 4 en el caso de DQPSK y 8 en el caso de D8PSK.

```
tx_bits = randi([0,1],l_trama,1);

% Aleatorización
[tx_aleatorio, vector_aleatorizacion] = aleatorizacion(tx_bits);

[x_ofdm, piloto] = mod_todo(2, tx_aleatorio, Nfft, Nofdm, Nf,0,0);

% Aplicación del canal
x_canal = filter(h, 1, x_ofdm);
x_ruido = awgn(x_canal, SNR_dB(snr)-offset_SNR,'measured');

% Demodulación
rx_bits = demod_todo(2, x_ruido, Nfft, Nofdm,
Nf,vector_aleatorizacion,0,0);
```

Las gráficas de las curvas BER vs SNR son las siguientes.

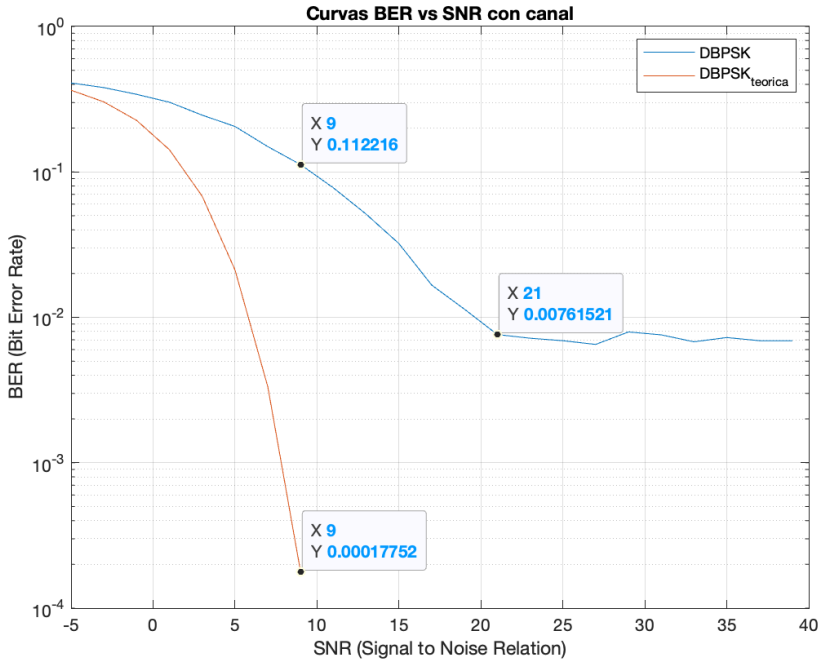

Ilustración 11 BER vs SNR con canal y sin ecualización DBPSK

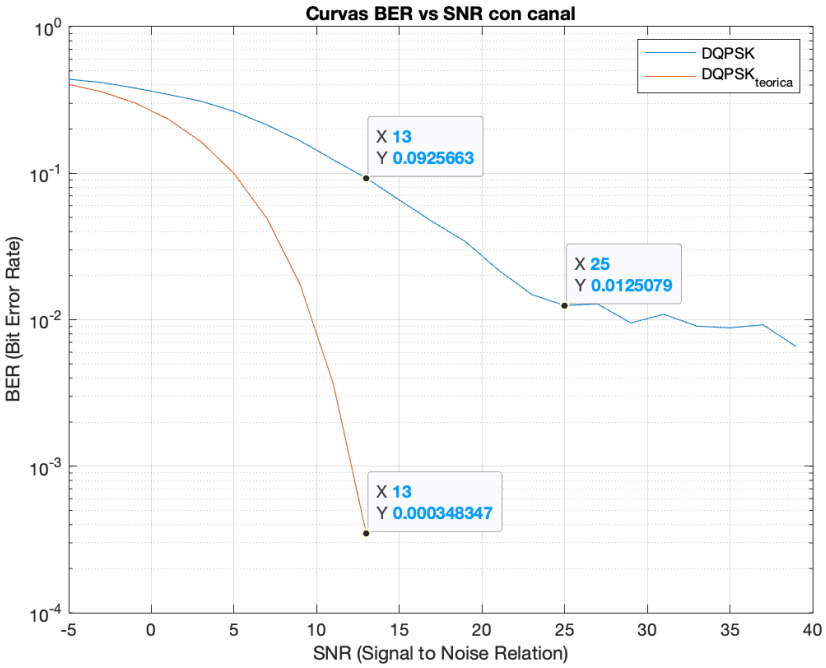

Ilustración 12 BER vs SNR con canal y sin ecualización DQPSK

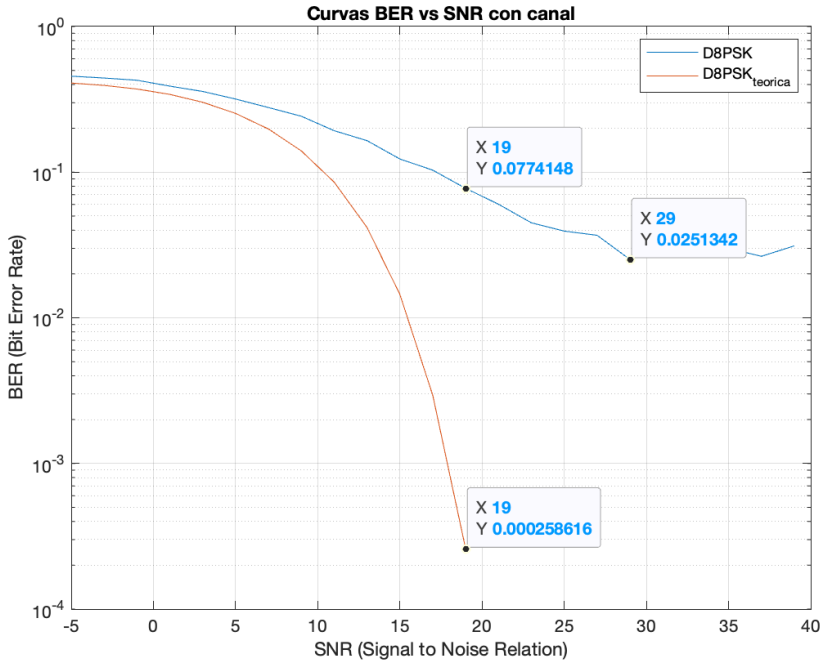

Ilustración 13 BER vs SNR con canal y sin ecualización D8PSK

Curvas BER vs SNR con prefijo cíclico y ecualización del canal

Para compensar los efectos del canal, se añade un prefijo cíclico (de duración superior al retardo máximo del canal) y se implementa un ecualizador en el dominio de la frecuencia después de la FFT. Además, se asume la disponibilidad de un primer símbolo OFDM piloto, cuyas amplitudes complejas (en cada subportadora) se conocen en el receptor y permiten estimar el canal. Esta información se utiliza para corregir la fase y la amplitud de las subportadoras en el resto de los símbolos, de modo que se reduce significativamente la interferencia y el error de demodulación. Al calcular y graficar las nuevas curvas de BER frente a la SNR, se observa una clara mejora en la detección de bits para los diferentes modos de transmisión sin FEC, acercándose de nuevo a la referencia ideal en canales AWGN y demostrando la eficacia de emplear prefijo cíclico y ecualización.

El prefijo cíclico se introduce para convertir la convolución lineal del canal en una circular desde la perspectiva del receptor, y así poder aplicar la transformada de Fourier de manera directa y sencilla para el proceso de ecualización. Sin el prefijo cíclico, la relación en frecuencia sería $X(k) \cdot H(k) \neq Y(k)$, siendo $X(k)$, $H(k)$, $Y(k)$ las trasformadas de $x[n]$, $h[n]$ e $y[n]$ respectivamente. Además, $y[n]$ es la señal muestreada de $y(t) = x(t) * h(t)$. Añadiendo un prefijo cíclico de longitud mayor o igual que la longitud del canal conseguimos obtener la convolución lineal a partir de la circular, logrando que $X(k) \cdot H(k) = Y(k)$. De esta forma, es posible estimar y compensar el canal dividiendo por $H(k)$ en receptor.

La longitud del canal es de 9 muestras, por lo que cualquier valor superior a 9 nos valdría como longitud del prefijo cíclico. Para nuestro sistema, hemos cogido el valor marcado en las especificaciones PRIME, el cual es de 48.

Cyclic Prefix (samples)	48

Ilustración 14 Tabla prefijo cíclico PRIME

A continuación, se muestra el código Matlab correspondiente a la función ecualización.

```
function out = ecualizacion(piloto,Nfft,Nofdm,Nf,x_ruido,Lcp)
x_rx = reshape(x_ruido, Nfft+Lcp, Nofdm);

% Eliminar prefijo cíclico
x_rx = x_rx(Lcp+1:end, :);

% Hago DFT
x = fft(x_rx,Nfft);

% Estimación del canal en funcnión del primer símbolo recibido
% (correspondiente al piloto).
w_i = piloto./x(88:88+Nf-1,1);

% Concateno coeficientes del canal
w = zeros(Nfft,Nofdm);
w(88:88+Nf-1,:) = repmat(w_i,1,Nofdm);

% Ecualizo
```

```
x_eq = x.*w;

% Vector fila
x_mod = x_eq(88:88+Nf-1,:);
out = reshape(x_mod,[],1);
end
```

Observamos que en las curvas BER vs SNR, el efecto de la ecualización ha eliminado por completo la estabilidad de la BER a partir de cierta SNR que observábamos en el apartado anterior. Si comparamos las mismas métricas que en el caso anterior, observamos que para la SNR que proporciona el último valor no nulo de BER teórica, la BER simulada ha aumentado considerablemente: 0,5 para DBPSK y 0,2 tanto para DQPSK como D8PSK.

En el caso de DBPSK, observamos que obtenemos el último valor de BER simulada no nulo con una SNR de $23dB$, mientras que para DQPSK y D8SPK es de $29dB$ y $33dB$ respectivamente. Esto concuerda con lo visto en teoría, ya que al tener DBPSK la mayor distancia entre símbolos, precisa de mucha menor SNR para obtener una mejor BER. D8PSK tiene la ventaja de poder empaquetar más información por constelación, pero es más débil frente a sistemas con mayor ruido.

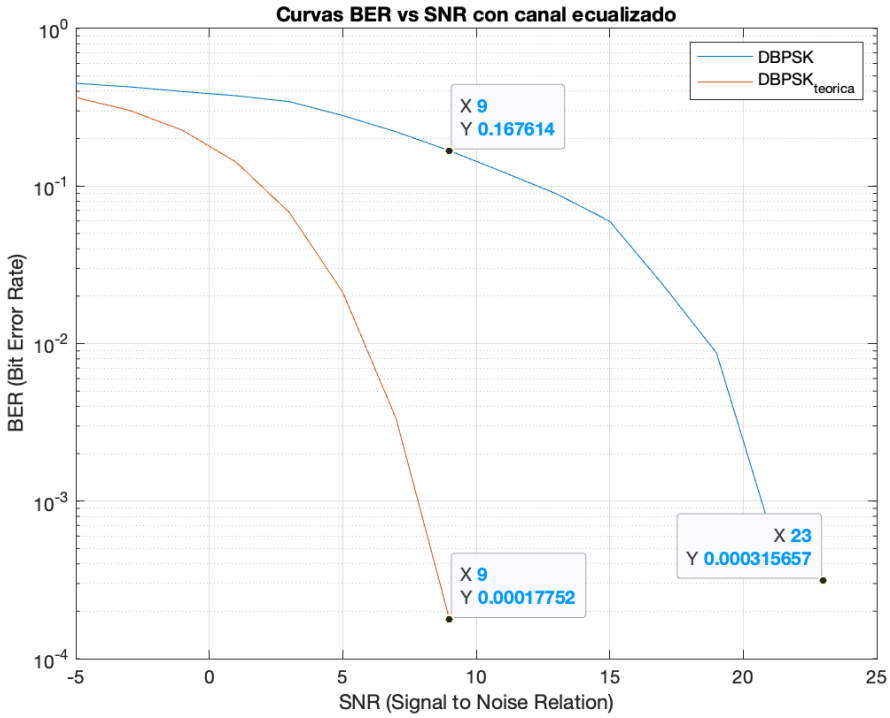

Ilustración 15 BER vs SNR canal ecualizado DBPSK

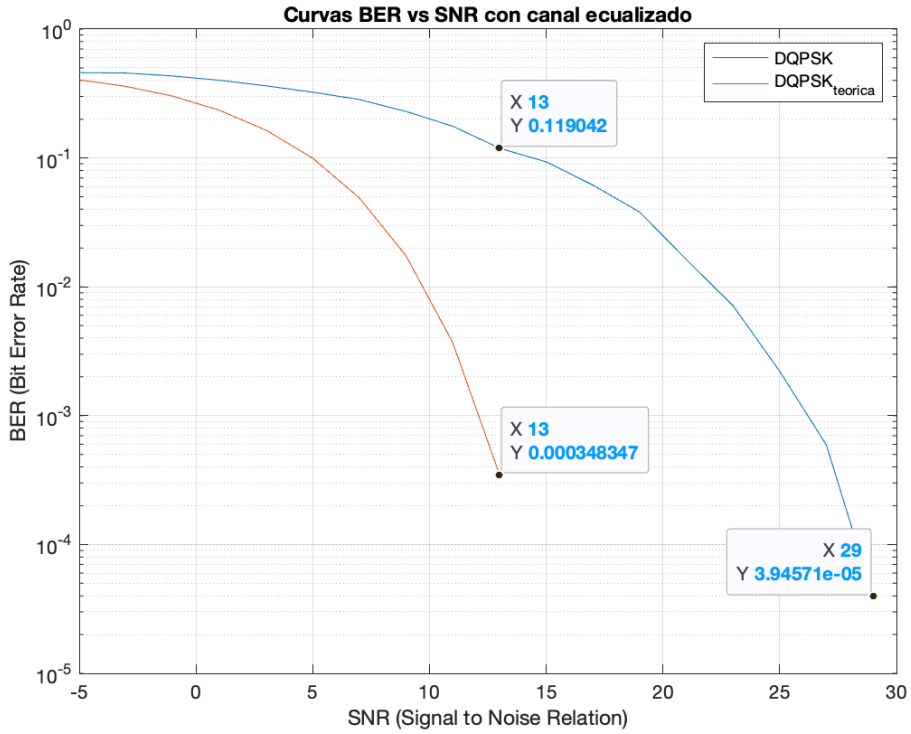

Ilustración 16 BER vs SNR canal ecualizado DQPSK

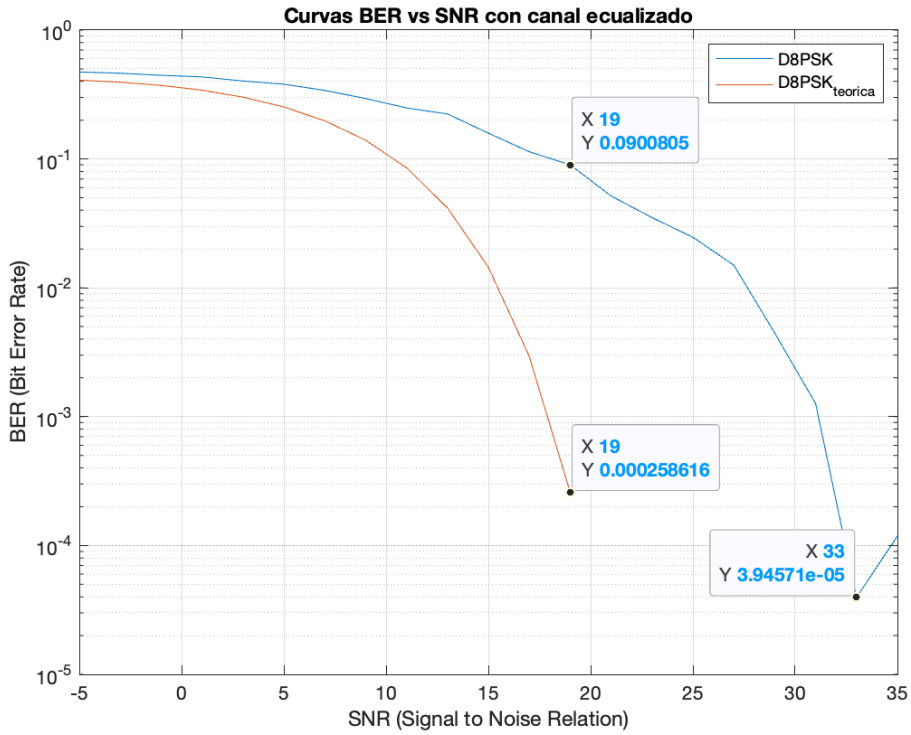

Ilustración 17 BER vs SNR canal ecualizado D8PSK

Conclusiones

Las conclusiones de este apartado muestran que la dispersión introducida por el canal con la respuesta al impulso $h[n]$ degrada la BER de forma apreciable cuando no se emplean técnicas de mitigación, pues la distorsión en frecuencia generada no puede compensarse con la demodulación diferencial por sí sola. En particular, se observa que la BER se mantiene en valores elevados, aunque se incremente la SNR. Con la inclusión de un prefijo cíclico de longitud superior al retardo máximo del canal y el uso de un ecualizador en frecuencia (estimado mediante un símbolo piloto), se consigue transformar la convolución circular en lineal, estimar la respuesta del canal y compensarla en el receptor. De este modo, las curvas BER vs SNR mejoran sustancialmente y se aproximan de nuevo al comportamiento teórico de un canal AWGN, poniendo de manifiesto la importancia de contar con prefijo cíclico y ecualización para comunicaciones OFDM en canales dispersivos.

Implementación de todos los modos de comunicación de PRIME con FEC

Objetivo

El objetivo de este apartado es incorporar y analizar el impacto de las técnicas de corrección de errores (FEC) en el sistema PRIME definido, evaluando cómo mejoran las prestaciones ante la presencia de un canal real y con ruido. Para ello, se ajustan los parámetros de simulación a fin de cumplir las especificaciones PRIME en cuanto a la generación de tramas codificadas (incluyendo bits de información y bits de relleno o flushing), se implementan y validan los módulos de entrelazado/desentrelazado y codificación/decodificación (con las funciones convenc y vitdec de MATLAB), y se comparan las nuevas curvas de BER frente a SNR con las previamente obtenidas sin FEC para los diferentes modos de modulación definidos en el estándar.

Modificación de parámetros de simulación

Para implementar FEC en la simulación, es necesario ajustar los parámetros de la trama OFDM. Con codificación convolucional de tasa 1/2, el número de bits de información transmitidos por subportadora se reduce a la mitad. La tabla adjunta muestra cómo, por ejemplo, en DBPSK se pasa de 96 a 48 bits de información por subportadora al activar la codificación, lo que repercute en el número total de bits de información por símbolo OFDM y, por consiguiente, obliga a modificar la construcción de las tramas payload para cumplir con las especificaciones PRIME. Además, la codificación convolucional requiere la introducción de bits de vaciado (flushing bits) para asegurar el vaciado de los registros del codificador tras el procesamiento de cada trama.

	DBPSK		DQPSK		D8PSK	
Convolutional Code (1/2)	On	Off	On	Off	On	Off
Information bits per subcarrier N_{BPSC}	0.5	1	1	2	1.5	3
Information bits per OFDM symbol N_{BPS}	48	96	96	192	144	288
Raw data rate (kbps approx)	21.4	42.9	42.9	85.7	64.3	128.6
Maximum MSDU length with 63 symbols (in bits)	3016	6048	6040	12096	9064	18144
Maximum MSDU length with 63 symbols (in bytes)"	377	756	755	1512	1133	2268

Ilustración 18 Tabla bits de información FEC ON

Debido a que nuestro sistema cuenta con una memoria de 7 posiciones (el bit de entrada y los 6 bits anteriores), crearemos un vector de vaciado o flush de 6 posiciones, para cada uno de los registros del codificador convolucional.

```
% Memoria del codificador
constraint_length = 7;

% Bits vaciado
flush_bits = zeros(constraint_length,1);
```

Inclusión de entrelazado en canal ideal

En ausencia de ruido (SNR muy alta), la incorporación de los bloques de entrelazado y desentrelazado no introduce errores adicionales en la transmisión, ya que su función principal es reordenar y restaurar la secuencia de bits sin alterarla. En la práctica, cuando se configura el sistema con un canal ideal (o con SNR teóricamente infinita), el receptor recupera los mismos bits que se transmitieron, confirmando que la tasa de error es cero. Así se verifica que el entrelazado y desentrelazado, por sí mismos, no degradan la señal ni modifican la información, sino que preparan la secuencia para enfrentar mejor los efectos de ráfagas de errores en condiciones de ruido real.

El objetivo del entrelazado es reordenar la secuencia de bits codificados de tal forma que, si el canal produce errores en ráfagas (también llamados errores burst), dichos errores se repartan a lo largo de la secuencia tras el desentrelazado en el receptor. Este mecanismo convierte los grupos consecutivos de bits erróneos en fallos distribuidos de uno o pocos bits por palabra codificada, que el decodificador puede corregir de manera más efectiva al contar con un código FEC que esté diseñado para manejar errores dispersos en lugar de ráfagas concentradas.

Según las especificaciones PRIME, debemos guardar un bloque de guarda igual al número de bits de información codificados, el cual sigue la siguiente expresión: $N_{CBPS} = 2 \cdot N_{BPS}$, donde $N_{BPS} \equiv$ número de bits de información. Para ello: (1) agruparemos los bits a transmitir en una matriz de dimensión $N_{símbolos} x N_{CBPS}$ y (2) transmitiremos las columnas de la matriz concatenadas.

En recepción, se realiza el proceso inverso: (1) agrupamos los bits recibidos en una matriz de dimensión $N_{CBPS} x N_{símbolos}$ y (2) nos quedamos con la secuencia resultante al

concatenar las columnas de la matriz. El código Matlab del entrelazado y desentrelazado es el siguiente.

```
function out = entrelazado(bits, Ncbps)
    bits_matrix = reshape(bits, length(bits)/Ncbps, Ncbps);
    out = reshape(bits_matrix', [], 1);
end

function out = desentrelazado(bits, Ncbps)
    bits_matrix = reshape(bits, Ncbps, length(bits)/Ncbps);
    out = reshape(bits_matrix', [], 1);
end
```

Comprobamos, que en ausencia de ruido en el canal, la transmisión funciona correctamente sin ningún error.

```
>> errores = sum(abs(bits_recibidos_concatenados-bits_transmitidos_concatenados))

errores =

     0
```

Inclusión FEC en canal ideal

En un escenario libre de ruido, se puede verificar que la cadena de transmisión y recepción que incluye el codificador y decodificador convolucional no introduce errores comparando los bits originales con los bits recuperados en el receptor. El diagrama de bloques mostrado ilustra un codificador convolucional de tasa 1/2, en el que cada bit de entrada pasa por un conjunto de retardos (Z^{-1}) y combinaciones lógicas que generan dos bits de salida. Esta estructura se define en MATLAB mediante la función poly2trellis, asignando los polinomios que describen el codificador (1111001 y 1011011, que corresponden con 171 y 133 en base octal y la longitud de memoria del codificador (constraint_length).

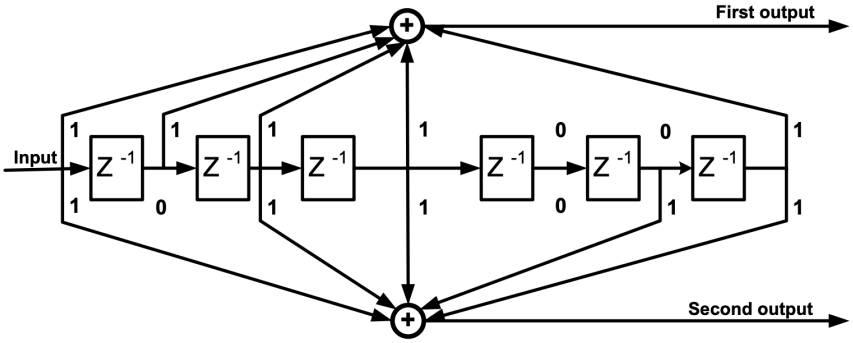

Ilustración 19 Representación digital código convolucional

Utilizamos las funciones convec para codificar la secuencia a transmitir y vitdec, la cual utiliza el algoritmo de viterbi para decodificar la secuencia. A esta última función es necesario pararle el traceback_length, el cual se define como el producto entre la memoria del codificador y el número de registros. En este caso: $7 \cdot 6 = 42$. El código Matlab empleado para la codificación y decodificación de tramas DBPSK es el siguiente. Para el

resto de tramas, únicamente es necesario cambiar el primer parámetro de las funciones mod_todo y comm.DPSKDemodulator por un 4 en el caso de DQPSK y 8 D8PSK.

```
% 171 y 133 son números en base octal correspondiente a 1111001 y
1011011
% respectivamente.
trellis = poly2trellis(constraint_length,[171 133]);
% Según sistema PRIME
traceback_length = 42;

tx_bits = randi([0,1],l_trama,1);

% Codifico bits
tx_codec = convenc([tx_bits; flush_bits], trellis);
tx_codec_flush = tx_codec(end-2*length(flush_bits)+1:end);
tx_codec = tx_codec(1:end-2*length(flush_bits));

% Aleatorización
[tx_aleatorio, vector_aleatorizacion] = aleatorizacion(tx_codec);

x_ofdm, piloto] = mod_todo(2, tx_aleatorio, Nfft, Nofdm, Nf,1,Lcp);

% Aplicación del canal
x_canal = filter(h, 1, x_ofdm);

% Ecualizacion
% Piloto del 87 al 87 + 96
x_eq = ecualizacion(piloto,Nfft,Nofdm,Nf,x_canal,Lcp);

% Demodulación
dbpsk_demod = comm.DPSKDemodulator(2,0,'BitOutput',true);
rx_bits_aleatorio = dbpsk_demod(x_eq);
rx_codec = desaleatorizacion(rx_bits_aleatorio,vector_aleatorizacion);

% Decodifico
rx_bits =
vitdec([rx_codec;tx_codec_flush],trellis,traceback_length,'trunc',
'hard');
rx_bits = rx_bits(1:end-length(flush_bits));
```

Una vez codificada la señal con la función convenc, si se transmite sin ruido por un canal ideal, el decodificador vitdec recupera exactamente la secuencia de bits original, demostrando que la codificación y decodificación funcionan sin generar errores en condiciones ideales.

```
>> errores = sum(abs(bits_recibidos_concatenados-bits_transmitidos_concatenados))

errores =

     0
```

Efecto del FEC en canal con dispersión

Durante este apartado analizaremos el efecto del codificador y corrección de errores en el sistema, dividiendo el apartado en 2 bloques: (1) sistema con FEC y (2) sistema sin FEC.

Sistema con FEC

Para la comparativa del FEC en las diferentes modulaciones utilizaremos dos métricas: (1) valor de BER simulado para la última SNR correspondiente a un BER teórico no nulo, (2) valor de la SNR para último BER simulado no nulo.

En primer lugar, empezamos con la modulación DBPSK para sistema con inclusión de FEC. Observamos que la última BER teórica no nula corresponde con una SNR de $9dB$. Esa SNR tiene una BER simulada de 0,20, un valor bastante alto, el cual indica que el 20% de los bits decodificados son erróneos. Sin embargo, observamos que la SNR correspondiente con el último valor BER simulada no nulo es de $19dB$, por lo que podemos considerar que para la modulación DBPSK, tenemos una BER = 0 para toda $SNR \geq 20dB$.

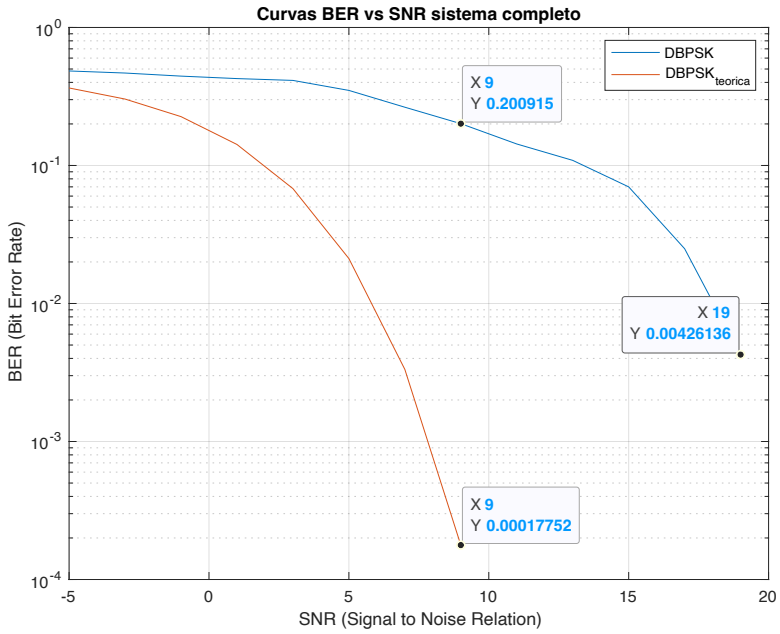

Ilustración 20 BER vs SNR sistema completo DBPSK

Para el caso de DQPSK, el último valor no nulo de BER teórica corresponde con SNR = $13dB$. Justo para esa SNR, la BER simulada es de 0,16, lo cual es menor que la simulada para la modulación DBPSK. Esto concuerda con los resultados obtenidos en el apartado anterior. La última SNR que presenta una BER no nula es de $25dB$, por lo que para la modulación DQPSK, podemos afirmar que toda $SNR \geq 26dB$ presentan una BER = 0.

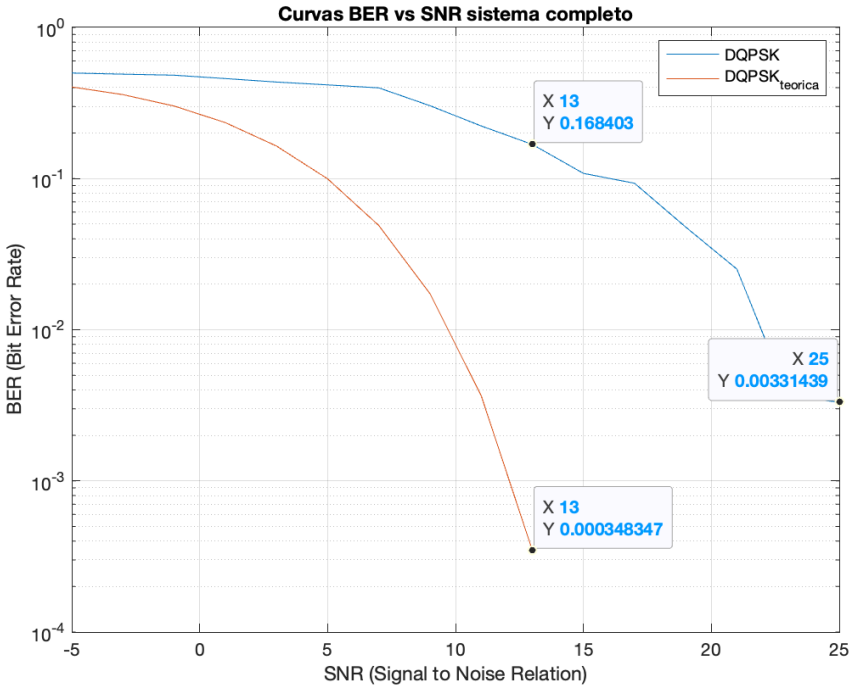

Ilustración 21 BER vs SNR sistema completo DQPSK

En el caso de la modulación D8PSK, la BER simulada para la última SNR proporciona un valor de BER teórica no nula vuelve a descender en comparación con la modulación anterior. En este caso, tiene un valor de 0,12, el cual sigue siendo demasiado alto de todas maneras. Observamos que la SNR que proporciona la última BER teórica no nula es la correspondiente a 29 dB, por lo que tendremos BER = 0 para toda $SNR \geq 30\ dB$.

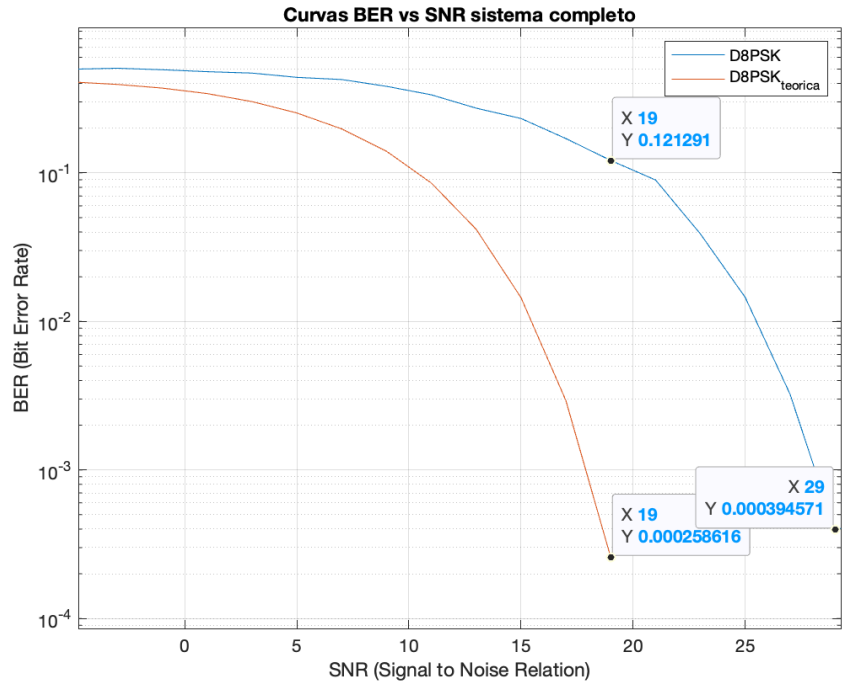

Ilustración 22 BER vs SNR sistema completo D8PSK

El código Matlab utilizado para la simulación de la curva de BER vs SNR para todas las modulaciones es el siguiente:

```
% Bucle que itera para todas las SNR
for snr=1:length(SNR_dB)
    % Bucle que itera para todas las modulaciones
    for i=1:3
        l_trama = L_tramas(i);
        n_tramas = N_tramas(i);

        bits_recibidos = zeros(n_tramas,l_trama);
        bits_transmitidos = zeros(n_tramas,l_trama);
        % Bucle que itera para todas las tramas
        for j=1:n_tramas
            % Si modulación de subportadora es DBPSK
            if i == 1
                tx_bits = randi([0,1],l_trama,1);

                % Codifico bits
                tx_codec = convenc([tx_bits; flush_bits], trellis);
                tx_codec_flush = tx_codec(end-
2*length(flush_bits)+1:end);
                tx_codec = tx_codec(1:end-2*length(flush_bits));

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_codec);

                % Entrelazado
                tx_entrelazado =
entrelazado(tx_aleatorio,2*l_trama/M);

                [x_ofdm, piloto] = mod_todo(2, tx_entrelazado, Nfft,
Nofdm, Nf,1,Lcp);

                % Aplicación del canal
                x_canal = filter(h, 1, x_ofdm);
                x_ruido = awgn(x_canal, SNR_dB(snr)-
offset_SNR,'measured');

                % Ecualizacion
                % Piloto del 87 al 87 + 96
                x_eq = ecualizacion(piloto,Nfft,Nofdm,Nf,x_ruido,Lcp);

                % Demodulación
                dbpsk_demod =
comm.DPSKDemodulator(2,0,'BitOutput',true);
                rx_entrelazado = dbpsk_demod(x_eq);
                rx_aleatorio =
desentrelazado(rx_entrelazado,2*l_trama/M);
                rx_codec = desaleatorizacion(rx_aleatorio,
vector_aleatorizacion);

                % Decodifico
                rx_bits = vitdec([rx_codec;
tx_codec_flush],trellis,traceback_length,'trunc', 'hard');
                rx_bits = rx_bits(1:end-length(flush_bits));

            % Si la modulación de subportadora es DQPSK
            elseif i == 2
                tx_bits = randi([0,1],l_trama,1);

                % Codifico bits
```

```
                tx_codec = convenc([tx_bits; flush_bits], trellis);
                tx_codec_flush = tx_codec(end-
2*length(flush_bits)+1:end);
                tx_codec = tx_codec(1:end-2*length(flush_bits));

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_codec);

                % Entrelazado
                tx_entrelazado =
entrelazado(tx_aleatorio,2*l_trama/M);

                [x_ofdm, piloto] = mod_todo(4, tx_entrelazado, Nfft,
Nofdm, Nf,1,Lcp);

                % Aplicación del canal
                x_canal = filter(h, 1, x_ofdm);
                x_ruido = awgn(x_canal, SNR_dB(snr)-
offset_SNR,'measured');

                % Ecualizacion
                x_eq = ecualizacion(piloto,Nfft,Nofdm,Nf,x_ruido,Lcp);

                % Demodulación
                dqpsk_demod =
comm.DPSKDemodulator(4,0,'BitOutput',true);
                rx_entrelazado = dqpsk_demod(x_eq);
                rx_aleatorio =
desentrelazado(rx_entrelazado,2*l_trama/M);
                rx_codec = desaleatorizacion(rx_aleatorio,
vector_aleatorizacion);

                % Decodifico
                rx_bits = vitdec([rx_codec;
tx_codec_flush],trellis,traceback_length,'trunc', 'hard');
                rx_bits = rx_bits(1:end-length(flush_bits));

            % Si la modulación de subportadora es D8PSK
            elseif i == 3
                tx_bits = randi([0,1],l_trama,1);

                % Codifico bits
                tx_codec = convenc([tx_bits; flush_bits], trellis);
                tx_codec_flush = tx_codec(end-
2*length(flush_bits)+1:end);
                tx_codec = tx_codec(1:end-2*length(flush_bits));

                % Aleatorización
                [tx_aleatorio, vector_aleatorizacion] =
aleatorizacion(tx_codec);

                % Entrelazado
                tx_entrelazado =
entrelazado(tx_aleatorio,2*l_trama/M);

                [x_ofdm, piloto] = mod_todo(8, tx_entrelazado, Nfft,
Nofdm, Nf,1,Lcp);

                % Aplicación del canal
```

```
                x_canal = filter(h, 1, x_ofdm);
                x_ruido = awgn(x_canal, SNR_dB(snr)-
offset_SNR,'measured');

                % Ecualizacion
                x_eq = ecualizacion(piloto,Nfft,Nofdm,Nf,x_ruido,Lcp);

                % Demodulación
                d8psk_demod =
comm.DPSKDemodulator(8,0,'BitOutput',true);
                rx_entrelazado = d8psk_demod(x_eq);
                rx_aleatorio =
desentrelazado(rx_entrelazado,2*l_trama/M);
                rx_codec = desaleatorizacion(rx_aleatorio,
vector_aleatorizacion);

                % Decodifico
                rx_bits = vitdec([rx_codec;
tx_codec_flush],trellis,traceback_length,'trunc', 'hard');
                rx_bits = rx_bits(1:end-length(flush_bits));

            end
            % Guardo tramas recibidas en una matriz
            bits_recibidos(j,:) = rx_bits;
            bits_transmitidos(j,:) = tx_bits;
        end
        % Concateno filas de la matriz
        bits_recibidos_concatenados =
reshape(bits_recibidos,1,N_bits_totales);
        bits_transmitidos_concatenados =
reshape(bits_transmitidos,1,N_bits_totales);

        % Cálculo BER para todas las tramas de la modulación
        BER(i,snr) = sum(abs(bits_recibidos_concatenados-
bits_transmitidos_concatenados))/N_bits_totales;
        %BER(BER<1e-5) = NaN;
    end
end

%% Representación BER vs SNR sistema completo
figure
semilogy(SNR_dB, BER(1,:))
hold on
semilogy(SNR_dB, BER_teor(1,:))
grid
legend('DBPSK','DBPSK_{teorica}')
ylabel('BER (Bit Error Rate)')
xlabel('SNR (Signal to Noise Relation)')
title('Curvas BER vs SNR sistema completo')

figure
semilogy(SNR_dB, BER(2,:))
hold on
semilogy(SNR_dB, BER_teor(2,:))
grid
legend('DQPSK','DQPSK_{teorica}')
ylabel('BER (Bit Error Rate)')
xlabel('SNR (Signal to Noise Relation)')
title('Curvas BER vs SNR sistema completo')
```

```
figure
semilogy(SNR_dB, BER(3,:))
hold on
semilogy(SNR_dB, BER_teor(3,:))
grid
legend('D8PSK','D8PSK_{teorica}')
ylabel('BER (Bit Error Rate)')
xlabel('SNR (Signal to Noise Relation)')
title('Curvas BER vs SNR sistema completo')
```

Sistema sin FEC

A continuación, realizaremos el mismo estudio que para el sistema completo con FEC, centrándonos en las 3 métricas descritas previamente.

Para el caso de DBPSK, la BER simulada correspondiente a la última BER teórica no nula es de 0,17, el cual es menor que en el sistema con FEC. Sin embargo, necesitamos una $SNR \geq 24dB$ para conseguir BER = 0. En el ejemplo anterior utilizando FEC, para la modulación DBPSK, esto se conseguía con una $SNR \geq 20dB$.

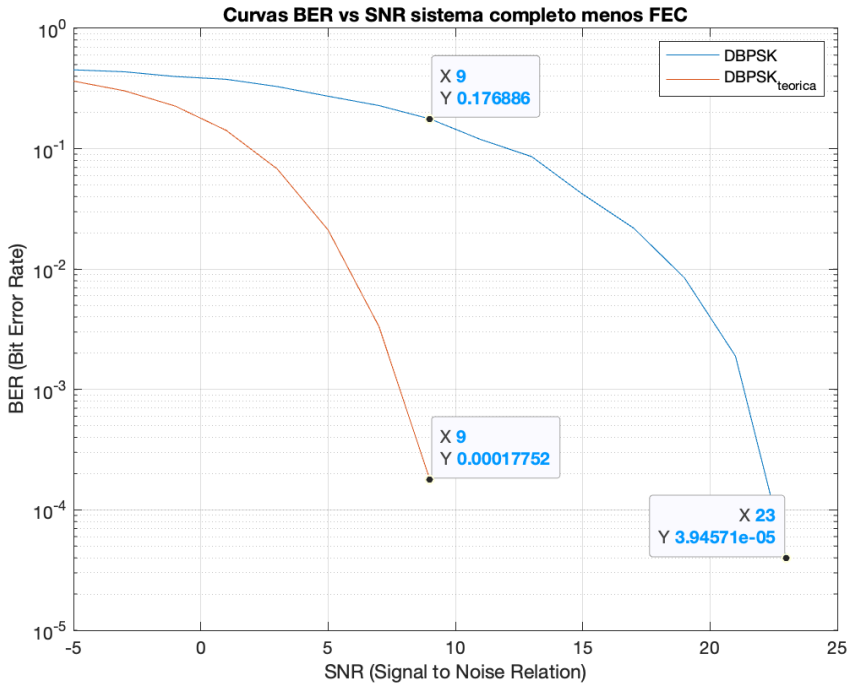

Ilustración 23 BER vs SNR sistema completo menos FEC DBPSK

Observamos el mismo patrón para la modulación DQPSK. El valor de BER simulada correspondiente a la última BER teórica no nula es de 0,13, menor que el 0,17 del caso con FEC. Sin embargo, vovlemos a comprobar que necesitamos de una SNR mayor para conseguir BER nulas. En este caso, obtendremos una BER = 0 para toda $SNR \geq 30dB$. En el caso con FEC, esta SNR era de apenas $25dB$, lo que es una diferencia significativa.

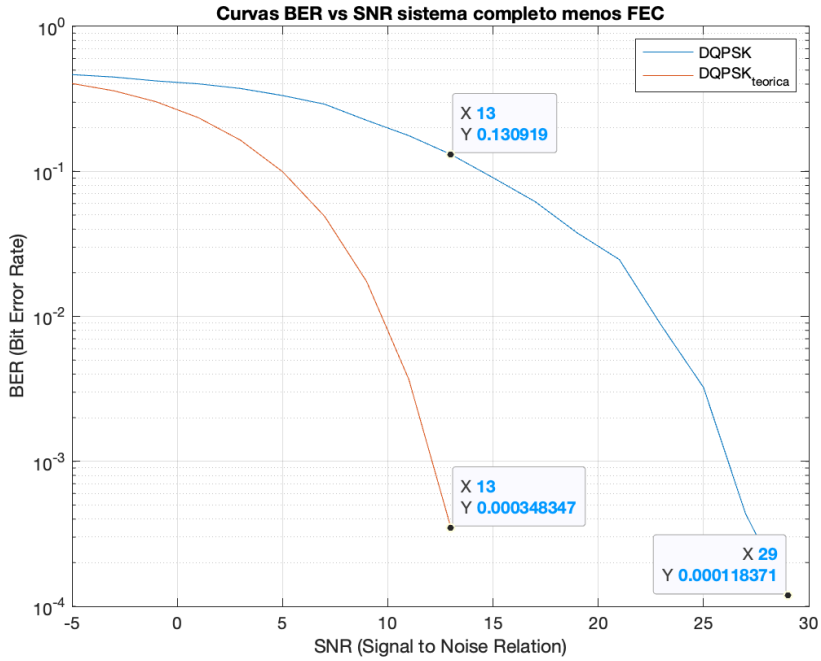

Ilustración 24 BER vs SNR sistema completo menos FEC DQPSK

Por último, analizamos el caso de D8PSK. La primera métrica sigue el patrón descrito previamente, el cual nos afirme que para casos con baja SNR, es mejor utilizar un sistema sin FEC. En este caso, la BER simulada correspondiente a la última BER teórica no nula es de 0,08 el cual se ve también disminuida en comparación con el 0,12 del caso con FEC. Sin embargo, si tenemos un sistema con una buena SNR, resulta recomendable utilizar corrección de errores o FEC. En el caso de D8PSK, la SNR correspondiente a la última BER no nula es de 35 dB, lo que es mucho mayor que los $29dB$ obtenidos en el caso con FEC.

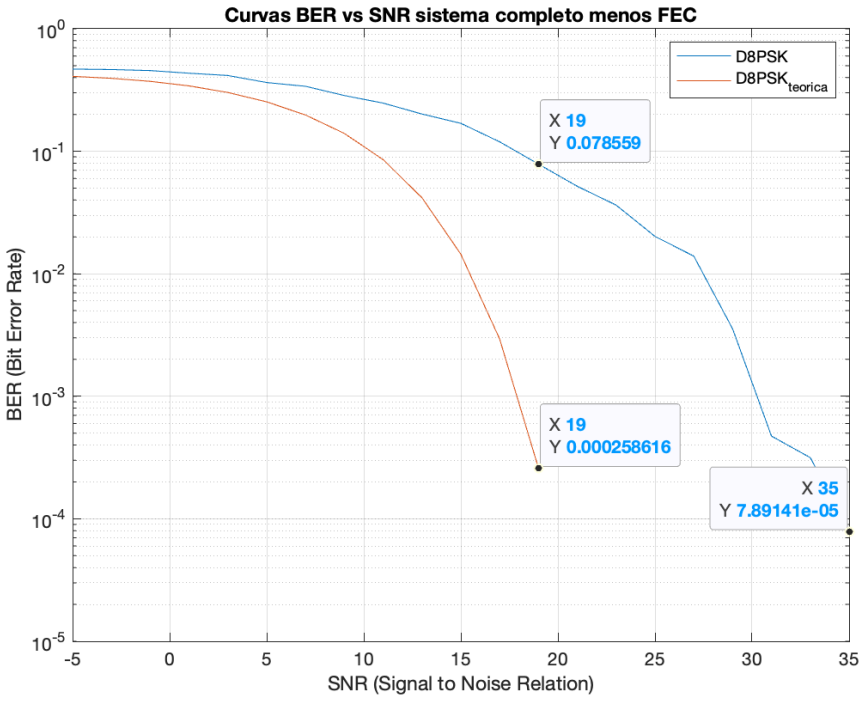

Ilustración 25 BER vs SNR sistema completo menos FEC D8PSK

Conclusiones

En términos generales, la incorporación de FEC reduce de manera significativa la SNR requerida para conseguir BER nula, especialmente cuando las condiciones de canal son razonablemente buenas. Por ejemplo, en DBPSK se pasa de necesitar 24 dB para alcanzar BER = 0 sin FEC a tan solo 20 dB con FEC. Sin embargo, en condiciones de baja SNR, el overhead de la codificación convolucional puede traducirse en BER simuladas más altas que en el sistema sin FEC, tal y como se observa en las comparaciones realizadas. Esto se ve reflejado en que, al evaluar la BER en la última SNR teórica no nula, resulta habitualmente algo más baja sin FEC, aunque se acabe necesitando una SNR mayor para conseguir BER nula.

En conjunto, los resultados muestran que, para enlaces con suficiente margen de SNR, la corrección de errores mejora notablemente la robustez de la transmisión, ya que permite alcanzar antes un régimen sin errores para DBPSK, DQPSK y D8PSK. Sin embargo, si la potencia de transmisión o la relación señal a ruido son limitadas, el uso de FEC introduce sobrecarga en los bits transmitidos y puede llevar a peores BER en el rango de SNR bajas. Por ello, la decisión de usar FEC o no depende tanto de la constelación elegida como de la disponibilidad de margen de SNR en el canal.