Control de errores

Procesamiento de señal en sistemas de comunicaciones y audiovisuales

Máster Universitario en Ingeniería de Telecomunicación

ETSIT-UPV

M.ª Á. Simarro Haro
A. González Salvador
F. J. Martínez Zaldívar

Índice

1. Introducción y objetivos

El control de errores o codificación de canal es un proceso esencial presente en prácticamente todo sistema de comunicación digital que pretenda aprovechar al máximo la *capacidad* del canal.

La *capacidad* de un canal se define como *la máxima información mutua que puede existir entre la entrada y la salida del mismo*. También podemos interpretarla como la máxima cantidad de información que puede transcurrir a través de dicho canal, libre de errores.

Dependiendo del modelo de canal que estemos empleando, podemos llegar a deducir distintas expresiones para el cálculo de la capacidad. Así, por ejemplo, si tenemos un canal binario y simétrico (*Binary Symmetric Channel* o BSC), su expresión es:

$$C = 1 - \mathrm{H}_2(p),$$

donde p es la probabilidad de error de bit del canal BSC y $\mathrm{H}_2(\cdot)$ es la función entropía binaria[1]. Sus unidades son bits de información por *uso* del canal.

Si el canal es continuo, por ejemplo de tipo AWGN (Additive White Gaussian Noise), la capacidad puede expresarse como:

$$C = W \log_2 \left(1 + \frac{S}{N}\right),$$

donde W representa el ancho de banda del canal en Hz, y el cociente S/N la relación señal-ruido (en escala lineal) en el mismo, por lo que las unidades serán bit/s (libres de errores).

Para poder aprovechar al máximo la capacidad del canal se requiere *codificación*. En su concepción más sencilla (figura 1), la codificación de canal consiste en establecer una correspondencia ente grupos de k bits (denominados mensajes) a la entrada del codificador de canal y grupos de n bits (denominados palabras código) a la salida del mismo.

Figura 1: Codificador de canal

[1] $$\mathrm{H}_2(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$$

El teorema de Shannon sobre la capacidad del canal nos decía que para que se pueda llevar a cabo la comunicación con una probabilidad de error arbitrariamente pequeña, es decir, tan pequeña como se desee, hay que llevar a cabo una codificación que respete que la ratio $R = k/n$ de la misma sea inferior a la capacidad de dicho canal; ello también suele implicar el trabajar con valores de n muy altos, lo que complica la codificación y sobre todo la decodificación desde un punto de vista práctico. Shannon explicó esta relación obligatoria que se debe cumplir para conseguir esa probabilidad de error arbitrariamente pequeña, pero no indicó cómo construir estos códigos. Todavía es un tema candente de investigación, más de medio siglo después de su enunciación.

1.1. Objetivos

En la presente práctica nos planteamos como objetivo analizar la probabilidad de error de bit residual tras realizar la decodificación de canal en el receptor, comprobando la no infabilidad de los códigos de corrección de errores, debido a sus limitaciones. Para ello, emplearemos distintos códigos utilizando como herramienta de simulación Matlab. De esta forma, conoceremos cómo Matlab permite el uso de códigos de canal para simular esta parte tan importante de un sistema de comunicación digital.

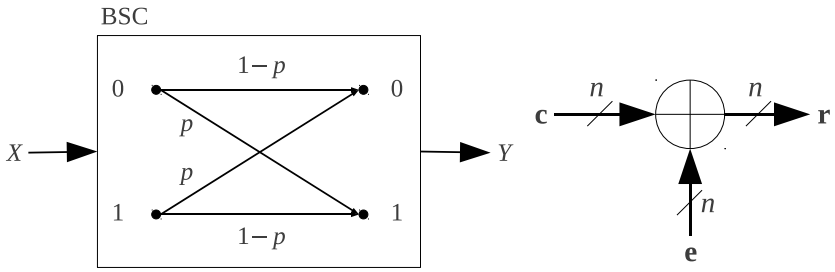
En términos generales, no podemos obtener una probabilidad de error idéntica a cero, pero sí tan pequeña como en principio quisiéramos. Para ello deberían cumplirse algunas hipótesis que hemos comentado en la subsección anterior. Además, algunas de esas hipótesis son difíciles llevarlas en la práctica, como por ejemplo disponer de un valor de n extremadamente alto, lo cual, como ya comentamos, complicaría especialmente la decodificación (y también la codificación). Todo ello, junto con el empleo de códigos no ideales redunda en que, desde un punto de vista práctico, no obtengamos una probabilidad de error *residual* idealmente nula, sino cierto valor que es el que pretendemos obtener en esta práctica.

Si pretendemos realizar una comparación equitativa entre un sistema con codificación de canal y otro sin ella, debemos realizar cierto reajuste *energético*. Si al introducir codificación de canal o control de errores no se modifican las características del sistema de modulación, estaríamos forzando a una disminución del régimen binario de la fuente, pero también a un incremento de la energía destinada al transporte de cada bit de información por un factor inversamente proporcional a la ratio de la codificación R, por lo que deberemos proceder a compensar este incremento en la energía si deseamos realizar estas comparaciones de manera ecuánime.

2. Modelo de canal

2.1. Modelo BSC

La figura 2 describe el modelo de canal BSC así como su posible simulación. El parámetro básico que define su comportamiento es la probabilidad de error de bit p, es decir, con qué probabilidad un 0 en la entrada se convierte en un 1 en la salida, o bien un 1 en la entrada se convierte en un 0 en la salida (y complementariamente, la probabilidad de no error de bit, $1-p$).



(a) Modelo BSC (b) Implementación

Figura 2: Modelo BSC e implmentación

A través del mismo podemos transmitir una secuencia de bits que potencialmente puede corromperse. Recordemos que podemos emular su comportamiento concibiendo la secuencia de salida **r** del mismo, como la de entrada **c** más (módulo-2) la secuencia que representa el denominado *patrón de error* **e**.

$$\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$$

El patrón de error viene marcado por la estadística del modelo BSC. Podemos simularlo en Matlab de la siguiente manera:

```
>> e = rand( 1, n ) < p;
```

de esta forma generaremos un patrón de error de n bits de forma que aparecerán *unos* con probabilidad p. (La función `rand` genera números reales aleatorios distribuidos uniformemente en el intervalo $r(0,1)$; si el real generado es inferior a p, la operación de comparación $< p$ dará cierto (que se traduce en un 1 en Matlab), en caso contrario dará falso que se traducirá como un 0). Por lo tanto

```
>> r = mod ( c + e, 2 );
```

representará la secuencia de bits recibida resultante de complementar sólo aquellos bits de la secuencia transmitida que coincidan con unos en las posiciones correspondientes del vector **e**. Ya que estos unos aparecen con probabilidad p, la probabilidad de error de bit al final es $P_b = p$. Como ejemplo, si

$$\begin{array}{rccccccc} & 1 & 1 & 0 & 0 & 1 & 0 & 1 & \equiv \mathbf{c} \\ \oplus & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \equiv \mathbf{e} \\ \hline & 1 & \mathbf{0} & 0 & \mathbf{1} & 1 & 0 & 1 & \equiv \mathbf{r} \end{array}$$

observamos cómo los bits segundo y cuarto comenzando por la izquierda llegan erróneos, tal y como así lo expresa el patrón de error **e**.

2.2. Relación entre p y E_b/N_0

La probabilidad de error a la salida del canal dependerá de qué características haya ocultado el modelo de canal: tipo de modulación, ancho de banda, relación señal a ruido S/N o E_b/N_0, etc.

Supongamos un caso sencillo en el que la modulación es de tipo binaria y polar (como BPSK, por ejemplo) y consecuentemente con probabilidad de error de bit:

$$p = P_b = \mathrm{Q}\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{1}$$

A estas alturas, podemos dibujar, ayudándonos de Matlab, la relación existente entre el parámetro E_b/N_0 (denominado en ocasiones como relación señal-ruido normalizada a bit) y la probabilidad de error de bit:

```
>> Eb_div_N0_dB = 0 : 1 : 10;
>> Eb_div_N0    = 10.^( Eb_div_N0_dB / 10 );
>> Pb           = Q( sqrt( 2 * Eb_div_N0 ) );
>> semilogy( Eb_div_N0_dB, Pb );
>> grid on
>> xlabel( 'Eb/N0 (dB)' );
>> ylabel( 'P_b' );
```

El alumno recordará que la función `semilogy` es equivalente a `plot` pero dibujando en escala logarítmica el eje de ordenadas. El alumno también habrá observado que la función de error `Q(x)` no está definida en Matlab, por lo que hay que diseñarla a partir de la función de error complementaria `erfc(x)`, que sí existe:

$$Q(x) = \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{2}$$

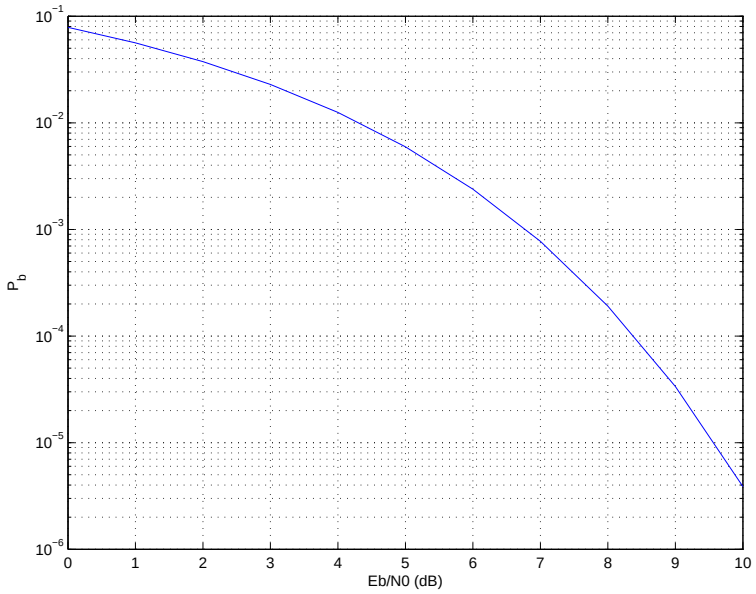


Figura 3: Probabilidad de error de bit en una modulación binaria polar

La figura 3 muestra el resultado del anterior *script* de Matlab.

2.3. Relación entre SNR y E_b/N_0

La relación señal ruido se define como

$$SNR = \frac{S}{N} = \frac{\frac{E_s}{T}}{N}$$

siendo S la potencia media de la señal, N la del ruido, E_s es la energía media por símbolo y T es el período de símbolo.

En ocasiones, las probabilidades de error no se dan en función de E_b/N_0 sino de la SNR. Para tal caso, a continuación se muestra la relación entre ambos.

N es la potencia del ruido que conviene calcular de la siguiente manera: el ruido es AWGN y se extiende en toda la banda de frecuencias, con una densidad espectral de potencia de $N_0/2$ W/Hz. La señal, al abandonar el canal e introducirse en el receptor, debe filtrarse dejando pasar exclusivamente lo que haya en la banda de interés: la propia señal y el ruido que esté inmerso en ese ancho de banda, tal y como indica la figura 4.

Entonces:

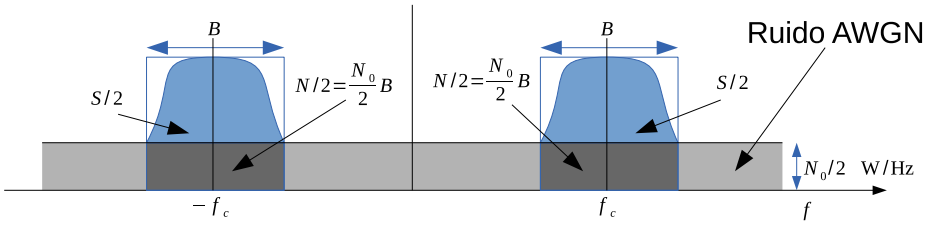


Figura 4: SNR en el canal

$$SNR = \frac{S}{N} = \frac{\frac{E_s}{T}}{N} = \frac{\frac{E_s}{T}}{N_0 B} = \frac{E_s}{N_0} \cdot \frac{1}{TB}$$

El ancho de banda mínimo necesario en paso-banda es de $B_{\min} = 1/T$ Hz según el criterio de Nyquist, lo cual proporcionará la máxima SNR coincidiendo numéricamente con E_s/N_0. Conforme aumentemos el ancho de banda del filtro a la entrada del receptor, irá disminuyendo la SNR, al dejar pasar más ruido para la misma señal. Por lo tanto:

$$SNR_{\max} = \frac{E_s}{N_0}$$

siendo la relación con E_b/N_0:

$$SNR_{\max} = \frac{E_b}{N_0} \log_2 M$$

En nuestro caso, las anteriores expresiones no serán necesarias, pues trabajaremos directamente con la relación señal-ruido normalizada a bit o E_b/N_0.

3. Códigos bloque

Un codificador bloque se denomina así por trabajar con bloques de información tanto a la entrada como a la salida del mismo: frente a un bloque de k bits a la entrada (mensaje), proporciona un bloque de n bits a la salida del mismo (palabra código); véase la figura 1.

Los parámetros genéricos más importantes de un código bloque derivados de n y k son la *ratio* $R = k/n$ y la *redundancia* $n - k$.

La codificación puede llevarse a cabo de muchas maneras; una de ellas es la matricial, en la que el mensaje es un vector de k bits, $\mathbf{m} \in \mathbb{B}^k$ y la palabra código correspondiente, uno de n bits, $\mathbf{c} \in \mathbb{B}^n$. La palabra código

se relaciona con su mensaje mediante la denominada matriz generadora del código $\mathbf{G} \in \mathbb{B}^{k \times n}$.

$$\mathbf{c} = \mathbf{mG}$$

Recuérdese que si el código pertenece a la versión sistemática del mismo, la matriz $\mathbf{G}$ tiene cierta estructura, siendo una posiblidad:

$$\mathbf{G} = \begin{pmatrix} \mathbf{I}_k & \mathbf{P} \end{pmatrix} \tag{3}$$

donde $\mathbf{I}_k$ es una matriz identidad de orden k y $\mathbf{P} \in \mathbb{B}^{k \times (n-k)}$ es la denominada matriz de paridad, por lo tanto:

$$\mathbf{c} = \mathbf{mG} = \mathbf{m} \begin{pmatrix} \mathbf{I}_k & \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{m} & \mathbf{mP} \end{pmatrix} = \begin{pmatrix} \mathbf{m} & \mathbf{p} \end{pmatrix} \tag{4}$$

siendo $\mathbf{p} \in \mathbb{B}^{n-k}$ los denominados *bits de paridad*.

La decodificación también puede llevarse a cabo de múltiples maneras. Una de ellas puede ser evaluar el denominado *sindrome* y tomar algún tipo de decisión a partir del valor del mismo. El síndrome se calcula a partir de la palabra recibida $\mathbf{r}$ (palabra código transmitida $\mathbf{c}$ y potencialmente alterada al atravesar el canal por un patrón de error $\mathbf{e}$), es decir, $\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$, y de la denominada matriz de comprobación de paridad $\mathbf{H}^T$ que para una codificación sistemática como la mostrada anteriormente, puede expresarse como:

$$\mathbf{H}^T = \begin{pmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{pmatrix}$$

donde $\mathbf{I}_{n-k}$ es la matriz identidad de orden $n-k$ y $\mathbf{P}$ es la matriz de paridad que apareció en la matriz generadora $\mathbf{G}$ en la forma sistemática del código, ecuación (3).

El síndrome $\mathbf{s} \in \mathbb{B}^{n-k}$ se calcula como

$$\mathbf{s} = \mathbf{rH}^T$$

Si dicho *síndrome* es nulo, la palabra recibida fue palabra código (aunque jamás se sabrá si fue la transmitida o no, pero lo más verosímil es que haya sido así) por lo que habrá que extraer el mensaje de dicha palabra código, lo cual es obvio a partir de (4). Si el síndrome es no nulo, denota que indiscutiblemente ha ocurrido algún tipo de error y ello obliga a ejecutar la estrategia oportuna de corrección del error. Como cabe esperar, cada código tiene una máxima capacidad de corrección de errores que se suele denotar con la letra t. Esta máxima capacidad depende de un parámetro de todo código bloque denominado *distancia mínima* o abrevidamente $d_{\min}$ con la siguiente expresión:

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

donde los símbolos $\lfloor \cdot \rfloor$ denotan el redondeo hacia menos infinito del argumento. Recuérdese que en un código bloque lineal, la distancia mínima del código es la menor cantidad de unos que nos encontramos en cualquier palabra código (exceptuando evidentemente la palabra código todo a ceros que siempre nos encontraremos en cualquier código bloque lineal).

3.1. Códigos Hamming

El nombre de estos códigos proviene del ideador de la versión Hamming(7,4), Richard Hamming (1950). Son un caso particular de códigos cíclicos con una capacidad antierror muy concreta: todos ellos tienen una distancia mínima de $d_{\min} = 3$ y por lo tanto, todos ellos pueden corregir hasta $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor = 1$ bit erróneo.

Los parámetros del código tienen el siguiente formato: n tiene la forma $n = 2^m - 1$, con $m \geq 3$ y $k = 2^m - 1 - m$, es decir, $n - k = m$. Esta última cantidad, $n - k = m$ es precisamente el grado del polinomio generador del código cíclico que genera el código Hamming, luego coincide con un polinomio primitivo de grado m.

Existe una manera alternativa de identificar a un código Hamming que consiste en diseñar su matriz de comprobación de paridad $\mathbf{H}^T \in \mathbb{B}^{n\times(n-k)}$ como el conjunto de todas las combinaciones de filas de $n-k$ bits excepto la combinación nula.

La forma de codificar, provocar errores y decodificar (corrigiéndolos) con códigos Hamming en Matlab es como sigue:

```
>> m = randi( [ 0 1 ], k, 1  ); % bits aleatorios equiprobables
>> c = encode( m, n, k, 'hamming/binary');
```

y la de introducir errores:

```
>> % Generacion de patron de bits con probabilidad p de que aparezcan unos
>> e = rand( size( c ) ) < p;
>> r = mod( c + e, 2 ); % palabra recibida
```

y la de decodificar (corrigiendo errores):

```
>> mDec = decode( r, n, k, 'hamming/binary' );
```

Si deseamos codificar y decodificar más de un mensaje, tendremos que encerrar las anteriores instrucciones en un bucle. Existe una alternativa que consiste en extender la longitud del array de bits, incluyendo un número entero de mensajes (por ejemplo 100000):

```
>> numMensajes = 100000;
>> mm = randi( [ 0 1 ], k * numMensajes, 1  );
>> cc = encode( mm, n, k, 'hamming/binary');
>> ee = rand( size( cc ) ) < p;
>> rr = mod( cc + ee, 2 ); % palabra recibida
>> mmDec = decode( rr, n, k, 'hamming/binary' );
```

3.2. Códigos BCH

La sigla proviene de los apellidos de sus ideadores: Raj Bose y D. K. Ray-Chaudhuri (1960) por una parte y Alexis Hocquenghem (1959) por otra, quienes los idearon de manera independiente. Podríamos concebirlos como una generalización de los códigos Hamming con la que podemos corregir más de un error.

En los códigos BCH, los parámetros n y k tampoco pueden ser cualesquiera. Concretamente, n debe tener la forma $n = 2^m - 1$, con $m \geq 3$. El parámetro t (cantidad de errores que puede corregir por palabra recibida o simplemente capacidad de corrección de errores del código BCH) se interrelaciona con los anteriores de la forma $n - k \geq mt$. No existe una fórmula cerrada que relacione los tres parámetros n, k y t. A continuación se tabulan algunas relaciones conocidas:

n	k	t
7	4	1

n	k	t
15	11	1
15	7	2
15	5	3

n	k	t
31	26	1
31	21	2
31	16	3
31	11	5
31	6	7

n	k	t
63	57	1
63	51	2
63	45	3
63	39	4
63	36	5
63	30	6
63	24	7
63	18	10
63	16	11
63	10	13
63	7	15

n	k	t
127	120	1
127	113	2
127	106	3
127	99	4
127	92	5
127	85	6
127	78	7
127	71	9
127	64	10
127	57	11
127	50	13
127	43	14
127	36	15
127	29	21
127	22	23
127	15	27
127	8	31

n	k	t
255	247	1
255	239	2
255	231	3
255	223	4
255	215	5
255	207	6
255	199	7
255	191	8
255	187	9
255	179	10
255	171	11
255	163	12
255	155	13
255	147	14
255	139	15
255	131	18
255	123	19
255	115	21
255	107	22
255	99	23
255	91	25
255	87	26
255	79	27
255	71	29
255	63	30
255	55	31
255	47	42
255	45	43
255	37	45
255	29	47
255	21	55
255	13	59
255	9	63

La forma de llevar a cabo la codificación y decodificación de códigos BCH en Matlab ha ido cambiando con las versiones. La que vamos a emplear en la presente práctica consiste en crear sendos objetos codificador y decodificador, en los cuales especificaremos exclusivamente los parámetros n y k, tal y como se muestra a continuación:

```
>> n = ...
>> k = ...
>> hEnc = comm.BCHEncoder( 'CodewordLength', n, 'MessageLength', k );
>> hDec = comm.BCHDecoder( 'CodewordLength', n, 'MessageLength', k );
```

y luego emplearlos en la rutina `step`:

```
>> numMensjes = 100000;
>> mm = randi( [ 0 1 ], numMensajes * k, 1 ); % mensajes de k bits
>> cc = step( hEnc, mm );  % palabras código
```

La transmisión y potencial corrupción en el canal sería, como en Hamming:

```
>> % Generacion de patron de bits con probabilidad p de que aparezcan unos
>> ee = rand( size( cc ) ) < p;
>> rr = mod( cc + ee, 2 ); % palabra recibida
```

La decodificación sería:

```
>> mmDec = step( hDec, rr );
```

3.3. Códigos RS

Sus siglas provienen de los apellidos de sus ideadores: Irving S. Reed y Gustave Solomon (MIT 1960). A su vez, los códigos RS podemos concebirlos como una generalización de los BCH pero para el caso no binario. Es decir, ahora los mensajes o palabras código no están compuestos por vectores de k bits o n bits respectivamente, sino de k ó n símbolos 2^m-arios respectivamente, donde cada símbolo podría interpretarse como una agrupación de m bits (cubriendo 2^m posibilidades). La notación que se suele emplear es que el símbolo en cuestión pertenece al cuerpo algebraico GF(2^m) —GF: Galois Field o cuerpo de Galois—.

Los parámetros fundamentales son m, n, k y t, y están relacionados de la siguiente manera:

$$\begin{aligned} m &\geq 3 \\ n &= 2^m - 1 \\ k &< n, \quad \text{con tal de que } n-k \text{ sea par} \\ t &= \frac{n-k}{2} \end{aligned}$$

En Matlab, el codificador y el decodicador se crean de manera similar a BCH:

```
hEnc = comm.RSEncoder( 'BitInput', true, 'CodewordLength', n, ...
                                         'MessageLength',  k );
hDec = comm.RSDecoder( 'BitInput', true, 'CodewordLength', n, ...
                                         'MessageLength',  k );
```

Se añade el par de parámetros `'BitInput', true` para que el codificador acepte bits, en vez de símbolos m-arios o pertenecientes al cuerpo GF(2^m).

Otra particularidad a tener en cuenta es entonces la cantidad de bits a generar: si queremos generar `nMensajes` y cada mensaje tendrá k símbolos, y cada símbolo se puede codificar con m bits (elemento de GF(2^m)), entonces:

```
>> nMes = 100000;
>> mm = randi( [ 0 1 ], nMensajes * k * m, 1 );
```

La codificacion y decodificación se llevan a cabo de manera similar a BCH, es decir, empleando la función `step` utilizando el objeto codificador o decodificador respectivamente.

3.4. Ganancia en la codificación

Es de suponer (al menos en cierto intervalo de relación señal-ruido) que obtendremos ventajas al emplear codificación de canal. La ventaja más obvia será obtener una probabilidad de error (residual) inferior a la situación sin codificar (incluso considerando la rectificación energética a la que hicimos mención en el subapartado 1.1). La ventaja podemos verla desde otro punto de vista: para obtener la misma probabildad de error de bit, necesitaremos menos E_b/N_0; la diferencia entre la E_b/N_0 sin codificación y la E_b/N_0 con codificación es la denominada *ganancia de codificación* para cierta probabilidad de error de bit.

4. Resultados

4.1. Parámetros de códigos

Dedúzcanse o búsquense los valores que faltan en la tabla 1 a excepción de la columna encabezada con G_c que se deberá rellenar al finalizar este apartado.

Código	m	n	k	R	t	G_c (dB)
Hamming	4					
Hamming	8					
BCH	4		5			
BCH	7		99			
RS	6				3	
RS	8				2	

Tabla 1: Códigos y sus parámetros de prueba

4.2. Curvas de probabilidades de error de bit

Para cada uno de los seis códigos de la tabla 1, obténganse las curvas de probabilidad de error de bit residual en función de E_b/N_0 (dB), tal y como se muestra en la figura 3, rectificando la energía debido a la ratio de la codificación.

Indicaciones y sugerencias (para cada código de la tabla 1):

- Trácense las curvas barriendo E_b/N_0 desde 0 hasta 10 dB con saltos de 1 dB y posteriormente pásese a escala lineal

```
>> Eb_div_N0_dB = 0 : 1 : 10;
>> Eb_div_N0 = 10.^( Eb_div_N0_dB / 10 );
```

- Diséñese la funcion `Q(x)` según la ecuación (2).

- Calcúlese la probabilidad de error del canal suponiendo una señalización binaria polar (BPSK, por ejemplo) a partir de cada valor de E_b/N_0 según la ecuación (1).

```
>> p_sin_cod = Q( sqrt( 2 * Eb_div_N0 ) );
```

  Estos valores obtenidos a partir de cada valor de E_b/N_0 formarán la curva de probabilidad de error de bit sin codificar que posteriormente se empleará de referencia para observar las bondades de la codificación. Es decir, debería obtenerse la curva que aparece en la figura 3.

- Para cada código según las especificaciones de la tabla 1, determinense los parámetros m, n, k, R y t,

```
>> m = ...
>> n = ...
>> k = ...
>> R = ...
>> t = ...
```

  - Corríjase E_b/N_0 y consecuentemente p teniendo en cuenta la ratio del código R:

```
>> p = Q( sqrt( 2 * R * Eb_div_N0 ) );
```

  - Genérense 100 000 mensajes aleatorios equiprobales de k bits

```
>> nMensajes = 100000;
>> mm = randi( [ 0, 1 ], k * nMensajes, 1 );
```

Si se trabaja con códigos RS en GF(2^m), recuérdese que la cantidad de bits a generar deberá ser `k * nMensajes * m` en vez de `k * nMensajes`. Además, se sugiere que para esta codificación sólo se simulen 10 000 mensajes para no extender demasiado el tiempo de simulación.

Para todos los códigos, se sugiere que se comience generando de momento sólo 1000 mensajes y conforme se vaya validando el software, se vaya incrementando dicha cantidad de mensajes para obtener una mayor resolución en los resultados.

- Codifíquense (empleando las llamadas oportunas a Matlab, según el código) los mensajes obteniendo las palabras código.
- Genérense los patrones de bits erróneos

```
>> patronesError = rand( size( palabrasCodigo ) ) < p;
```

- Prodúzcanse los errores:

```
>> palabrasRecibidas = mod( palabrasCodigo + patronesError, 2 );
```

- Decodifíquese (empleando las llamadas oportunas a Matlab, según el código) obteniendo los mensajes decodificados a partir de las palabras recibidas
- Llévese cuenta de cuántos bits erróneos se han producido tras la decodificación. Recuérdese que un código de control de errores posee una máxima capacidad de corrección; si la cantidad de errores producidos supera dicha capacidad, dejará de corregir *correctamente*. Sugerencia: súmense en módulo dos los mensajes decodificados con los del origen y acumúlense la cantidad de unos que se obtienen:

```
>> sum( mod( mensajes + mensajesDecodificados, 2 ) )
```

Ejemplo: sea m el mensaje transmitido y $mDec$ el mensaje decodificado (supuestamente corregido), entonces la cantidad de bits erróneos residuales será:

$$\begin{aligned} m &= 1011011 \\ mDec &= 101\mathbf{0}01\mathbf{0} \\ m \oplus mDec &= 000\mathbf{1}00\mathbf{1} \\ \text{sum}(m \oplus mDec) &= 2 \end{aligned}$$

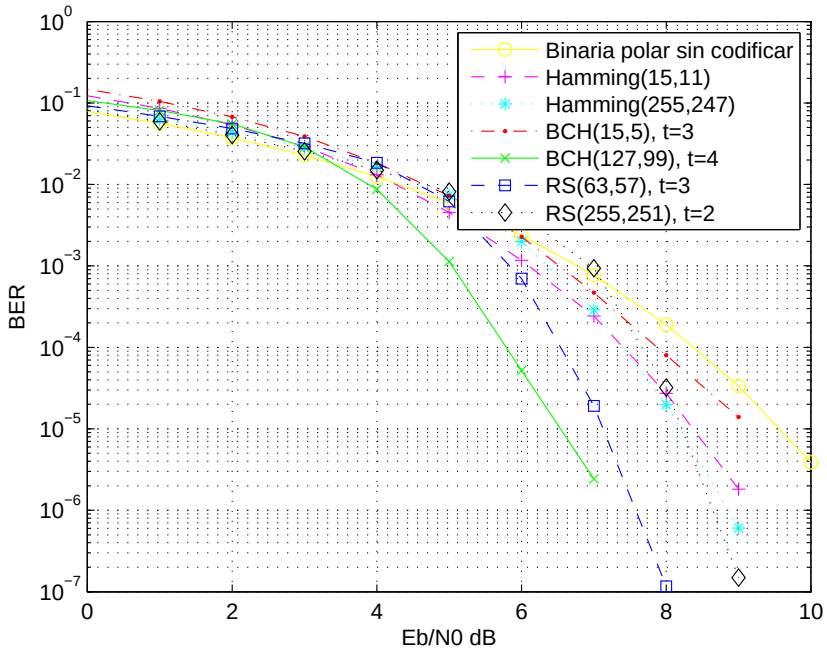


Figura 5: Probabilidad de error de bit con y sin codificación de canal

Esto lo podemos hacer, no sólo a nivel de mensaje, sino a nivel de secuencia de mensajes. En cualquier caso, si luego dividimos esta cantidad de bits erróneos por la cantidad total de bits transmitidos tendremos el denominado BER (Bit Error Ratio) que puede interpretarse como la *probabilidad de error de bit experimental*:

```
>> BER = sum( mod( mensajes + mensajesDecodificados, 2 ) ) ...
        / ( nMensajes * k );
```

Debe recordarse que para los códigos RS en GF(2^m), en princpio se debieron generar `nMensajes * k * m` bits en vez de `nMensajes * k` bits.

Si pintamos todas las curvas, de manera aproximada debe obtenerse el resultado que se muestra en el figura 5.

Obténgase estos resultados y compárense con el de la figura 5. Calcúlese sobre la gráfica la ganancia de codificación a una probabilidad de error de bit de $P_b = 10^{-4}$. Esta ganancia es el valor que debe introducirse en la columna encabezada por G_c de la tabla 1.

4.3. Resultados a entregar

Debe entregarse la tabla 1 rellenada, junto con la gráfica obtenida equivalente a la de la figura 5, así como el o los *script* empleados a tal efecto.