3. IMPLEMENTACIÓN DE ALGORITMOS DE DECODIFICACIÓN

En este capítulo estudiaremos simplificaciones de los algoritmos matemáticamente exactos que nos permitirán su implementación real para su uso en sistemas de comunicaciones. Veremos 2 simplificaciones del algoritmo LLR-BP y 2 para el enfoque de Gallager e iremos presentando las simulaciones realizadas respecto al algoritmo "ideal" (en realidad no es ideal ya que hemos truncado el argumento de la tanh pero las diferencias son despreciables) para estudiar la degradación sufrida. También presentaremos un análisis de la complejidad computacional orientada a su posible implementación hardware en el apéndice A.

Para la realización de las simulaciones hemos limitado el número de iteraciones a 50 por ser un valor muy usado en este tipo de estudios y hemos transmitido hasta un máximo de 10000 palabras de 64800bits, buscando 50 palabras erróneas y 1000 errores como mínimo. Las gráficas muestran cierta desviación en los valores, con lo que los resultados no son muy precisos numéricamente pero como ya hemos comentado, nuestro objetivo es realizar una comparación cualitativa entre los algoritmos. Las palabras de código han sido moduladas en BPSK y transmitidas por un canal AWGN.

3.1 Simplificaciones del algoritmo LLR-BP

Las simplificaciones consistirán en usar representaciones aproximadas de la tanh. Vamos a estudiar dos soluciones: una aproximación lineal a trozos (piecewise) y una aproximación constante a trozos mediante una LUT. Las estudiaremos por separado frente el algoritmo LLR-BP ideal y después las compararemos entre ellas.

3.1.1 Simplificación PW

La aproximación lineal PW aproxima una función continua usando distintos segmentos lineales en cada intervalo. Esto permite aproximar cada segmento de la curva usando una ecuación lineal en lugar de la ecuación de orden superior propia de la curva. Debemos dividir entonces la curva en segmentos, cuanto más pequeños sean, mejor será la aproximación. Cada uno de los segmentos vendrá definido por una ecuación $B(x) = ax + b$.

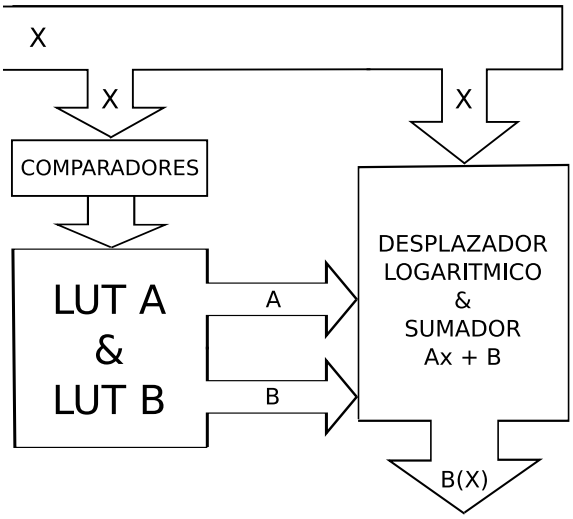

Figura 3.1: Implementación Hardware PW con registro de desplazamiento

Solo tendremos que almacenar las variables a y b en una pequeña LUT (una por cada variable) para su implementación en hardware. El resto de las operaciones hardware serán la suma y la multiplicación de la ecuación lineal $B(x)$. Se podrían tomar los valores de a como potencias de 2 para convertir el multiplicador en un registro de desplazamiento, simplificando aún más el hardware. Aunque esta solución es muy fácil de implementar en hardware, resulta lenta, ya que emplea una comparación (para determinar el segmento), un desplazamiento y una suma, por lo que se podría producir un cuello de botella. Además, si empleamos el registro de desplazamiento en lugar del multiplicador, aunque simplificamos el hardware, perdemos precisión, ya que estamos limitados por las potencias de 2 disponibles. En la figura 3.1 podemos ver su implementación hardware.

En este estudio hemos usado el multiplicador en lugar del registro de desplazamiento y se han seleccionado 7 regiones (tabla 3.1). Como la tanh es casi lineal en en la región $x \in [-0{,}8, 0{,}8]$ basta con 7 regiones en lugar de 8. Los valores de estas aproximaciones se encuentran por ensayo y error, basándose en el mínimo error cuadrático medio (RMSE) que en este caso es de aproximadamente 0.02 [28].

De una manera similar a la aproximación para la tanh, se propone una aproximación para la $\tanh^{-1}$. Basándonos en la correspondencia uno a uno entre las dos funciones, se propone una $\tanh^{-1}$ con 7 regiones (tabla 3.2).

Se puede reducir el número de elementos de estas tablas si tenemos en cuenta que la degradación del rendimiento es mínima para $x_0 = 3$ o 7 (tomando el valor de truncamiento del argumento de la tangente que usábamos para solucionar el problema de la precisión finita como $x_0 = 3$).

x	$\tanh(x)$
(-7.0, -3.0]	0.0012·x−0.9914
(-3.0, -1.6]	0.0524·x−0.8378
(-1.6, -0.8]	0.322·x−0.4064
(-0.8, 0.8]	0.83·x
(0.8, 1.6]	0.322·x+0.4064
(1.6, 3.0]	0.0524·x+0.8378
(3.0, 7.0]	0.0012·x+0.9914

Tabla 3.1: Aproximación a trozos de la tanh

x	$\tanh^{-1}(x)$
(-0.999998, -0.9951]	(x + 0.9914)/0.0012
(-0.9951, -0.9217]	(x + 0.8378)/0.0524
(-0.9217, -0.6640]	(x + 0.4064)/0.322
(-0.6640, 0.6640]	x/0.83
(0.6640, 0.9217]	(x - 0.4064)/0.322
(0.9217, 0.9951]	(x - 0.8378)/0.0524
(0.9951, 0.999998]	(x - 0.9914)/0.0012

Tabla 3.2: Aproximación a trozos de la atanh

En la figura 3.2 podemos ver la aproximación PW de la tanh y frente al valor ideal.

En las gráficas 3.3, 3.4, 3.5 y 3.6 podemos ver las BER obtenidas usando esta aproximación frente a la ideal.

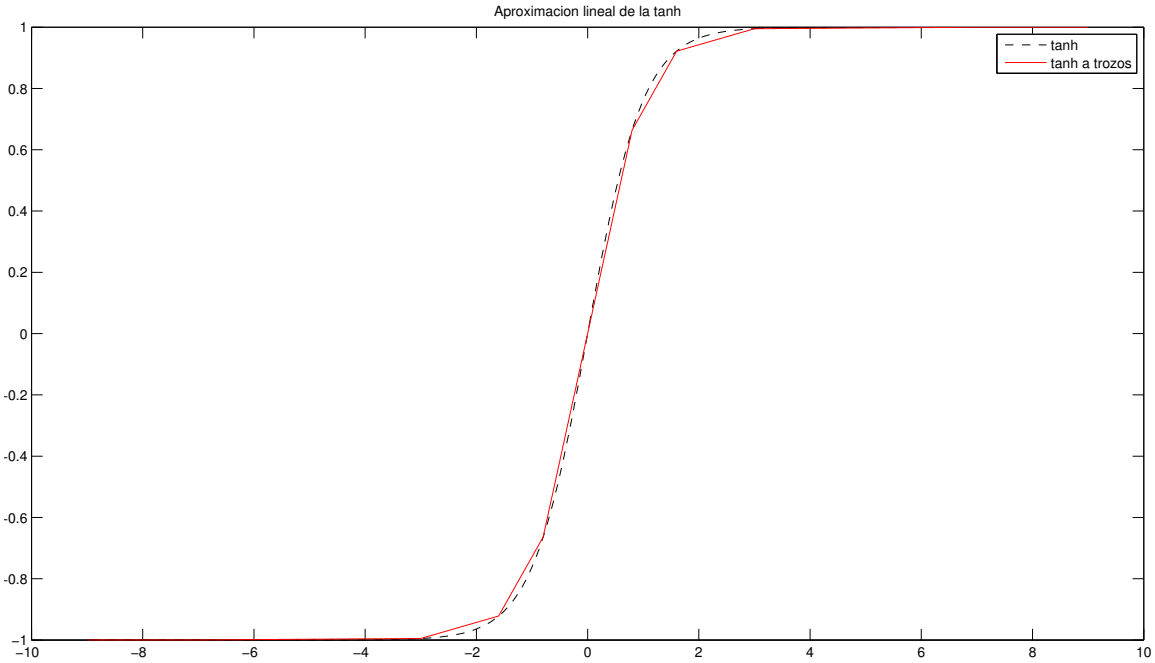

Figura 3.2: Tangente hiperbólica a trozos

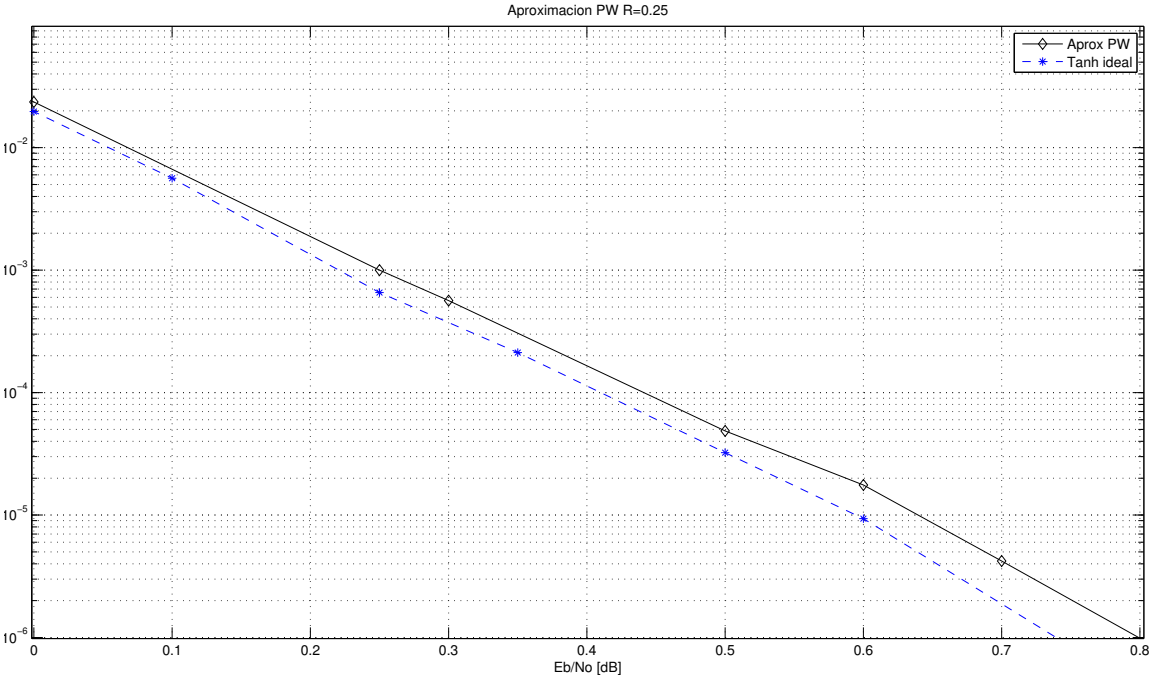

Figura 3.3: Aproximación PW R=0.25 Con la matriz de DVB-S2

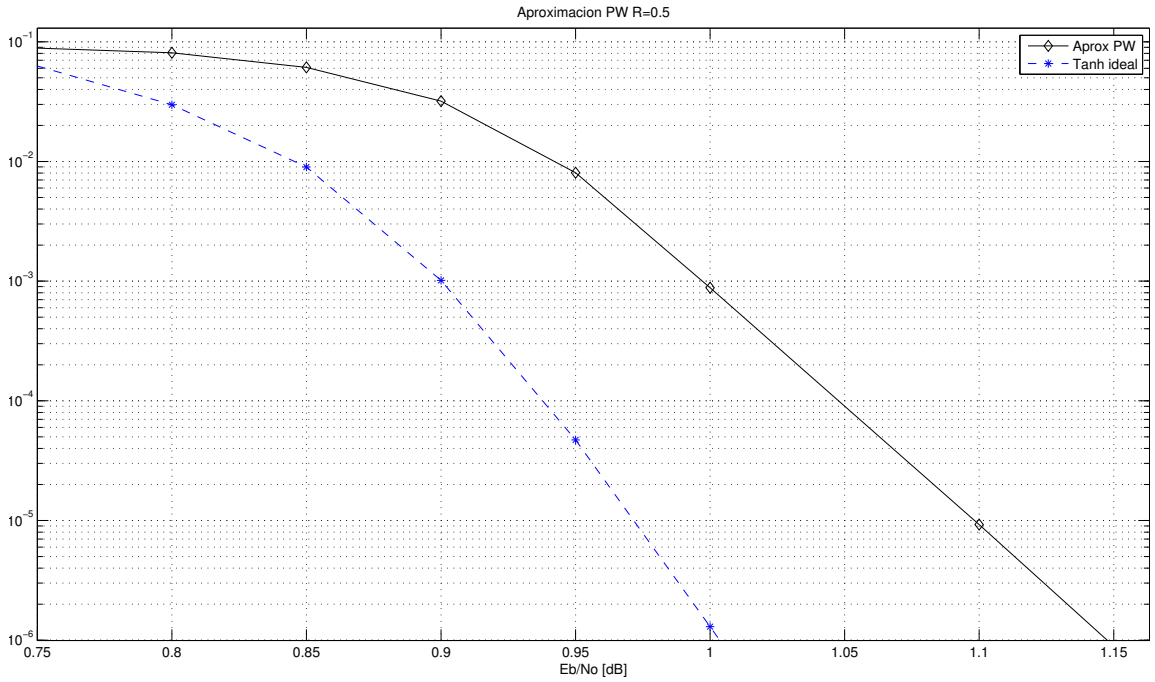

Figura 3.4: Aproximación PW R=0.5 Con la matriz de DVB-S2

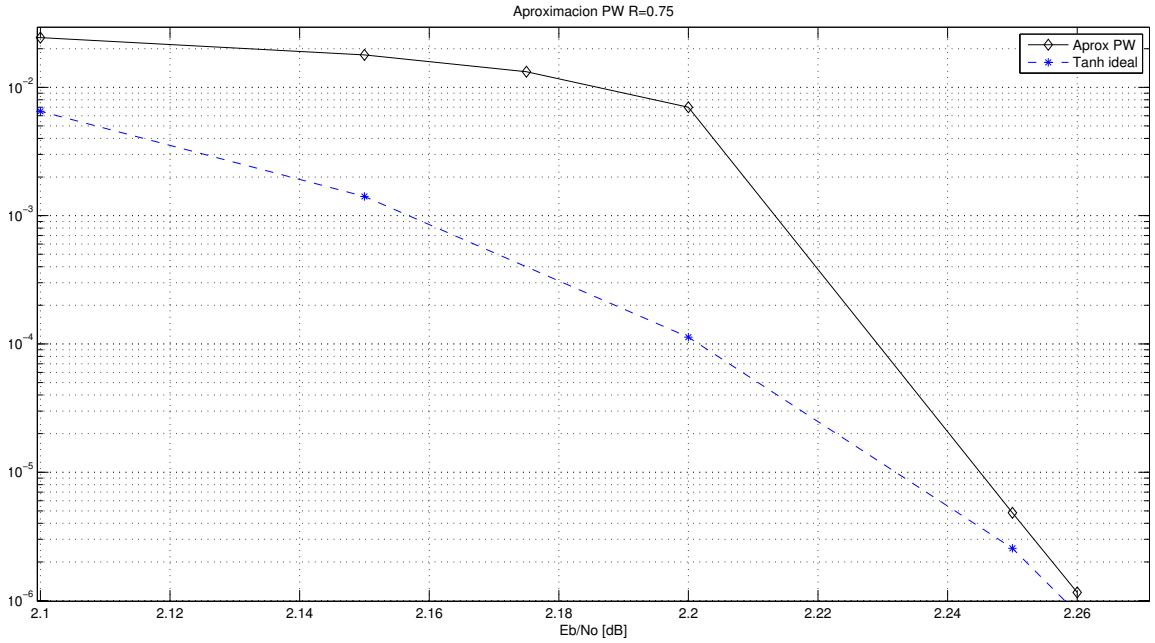

Figura 3.5: Aproximación PW R=0.75 Con la matriz de DVB-S2

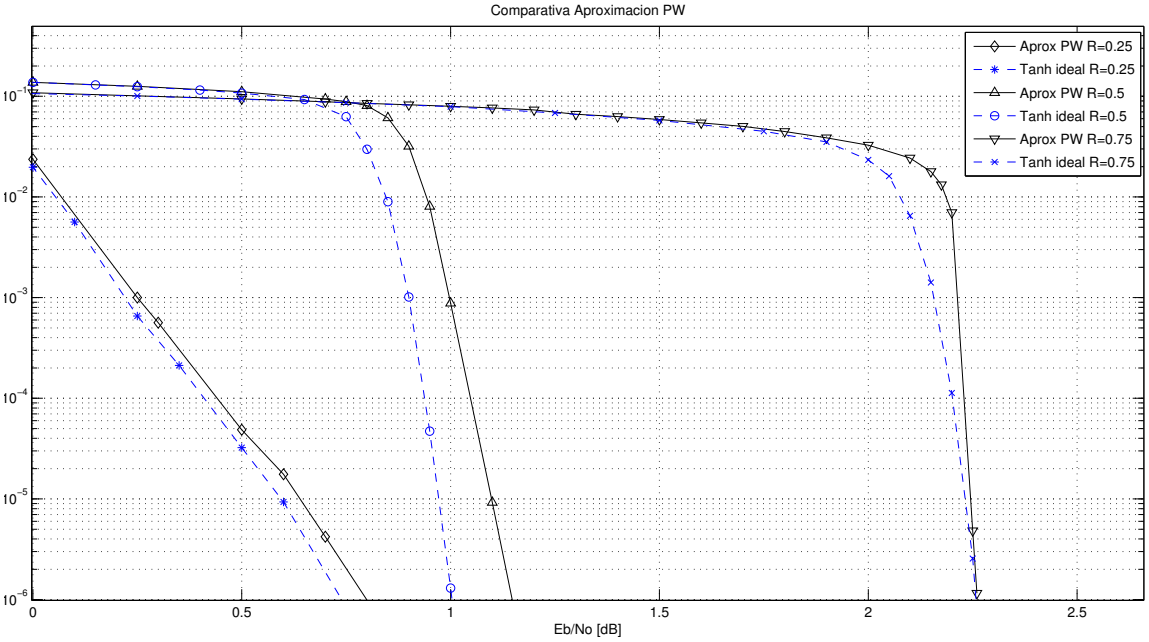

Figura 3.6: Aproximación PW, R variable con la matriz de DVB-S2

3.1.2 Simplificación LUT

Esta simplificación se base en el uso de una LUT guardada en una ROM para calcular las funciones hiperbólicas. El algoritmo es fácil, calcularemos los valores de la función con la precisión que determinemos y se guardarán en una tabla, haciendo corresponder los bits de x con la dirección del B(x) calculado. En teoría, esta solución podría presentar una precisión del 100 % para tamaños de la memoria grandes, pero como nos vemos limitados al número de elementos se obtendrá una precisión mucho menor. También el tamaño de las palabras de código implicará un tamaño mayor de la ROM (además de la lógica de direccionamiento).

En las simulaciones hemos usado un número de regiones similar al de la aproximación PW para poder comparar las dos soluciones. Para la tabla de cuantización se han usado 8 regiones, como se muestra en la tabla 3.3. Al igual que en el caso de la aproximación PW, los valores se encuentran por ensayo y error en base al RMSE, que en este caso es de 0.007.

De una manera similar a la aproximación para la tanh, se propone una aproximación para la $\tanh^{-1}$. Basándonos en la correspondencia uno a uno entre las dos funciones, se propone una tabla de cuantificación con 8 valores para la $\tanh^{-1}$ (tabla 3.4).

x	$\tanh(x)$
(-7.0, -3.0]	-0.99991
(-3.0, -1.6]	-0.9801
(-1.6, -0.8]	-0.8337
(-0.8, 0]	-0.3799
(0, 0.8]	0.3799
(0.8, 1.6]	0.8337
(1.6, 3.0]	0.9801
(3.0, 7.0]	0.99991

Tabla 3.3: Tabla de cuantización de la tanh

x	$\tanh^{-1}(x)$
(-0.999998, -0.9951]	-3.3516
(-0.9951, -0.9217]	-1.9259
(-0.9217, -0.6640]	-1.0791
(-0.6640, 0.0]	-0.3451
(0.0, 0.6640]	0.3451
(0.6640, 0.9217]	1.0791
(0.9217, 0.9951]	1.9259
(0.9951, 0.999998]	3.3516

Tabla 3.4: Tabla de cuantización de la atanh

En esta aproximación también podemos reducir el número de elementos si usamos $x_0 = 3$.

En la figura 3.8 podemos ver la aproximación LUT de la tanh y frente al valor ideal y en la figura 3.7 podemos ver la comparación entre esta aproximación y la anterior. En las gráficas 3.9, 3.10, 3.11 y 3.12 podemos ver las BER obtenidas usando esta aproximación frente a la ideal.

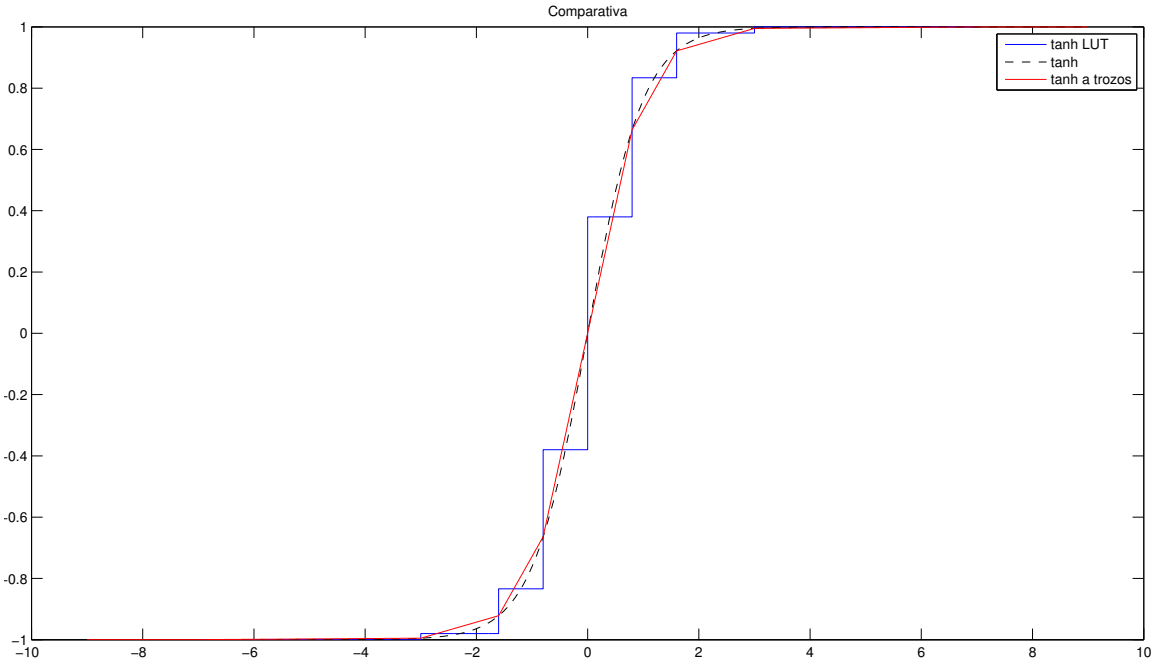

Figura 3.7: Comparativa

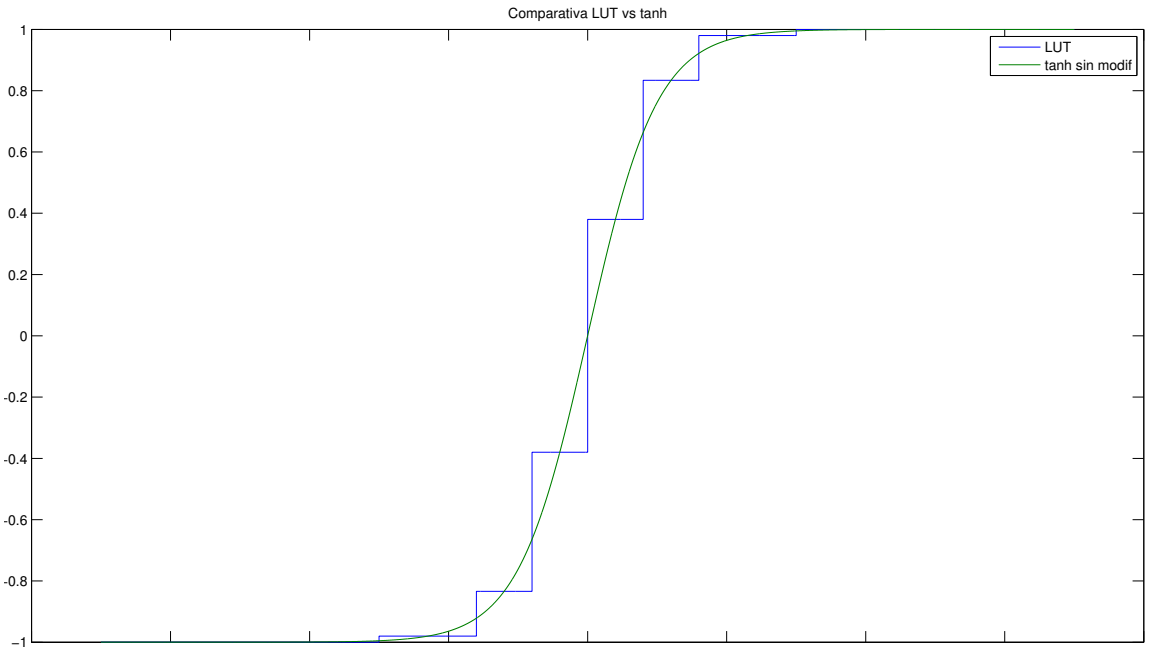

Figura 3.8: Tangente hiperbólica cuantizada

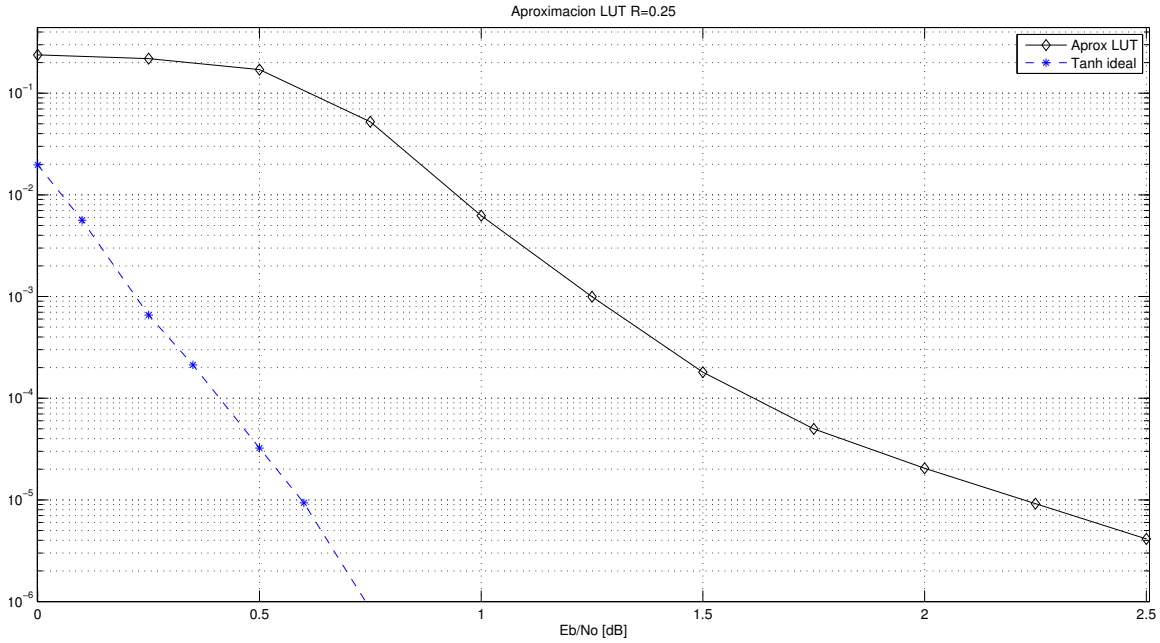

Figura 3.9: Aproximación LUT R=0.25 Con la matriz de DVB-S2

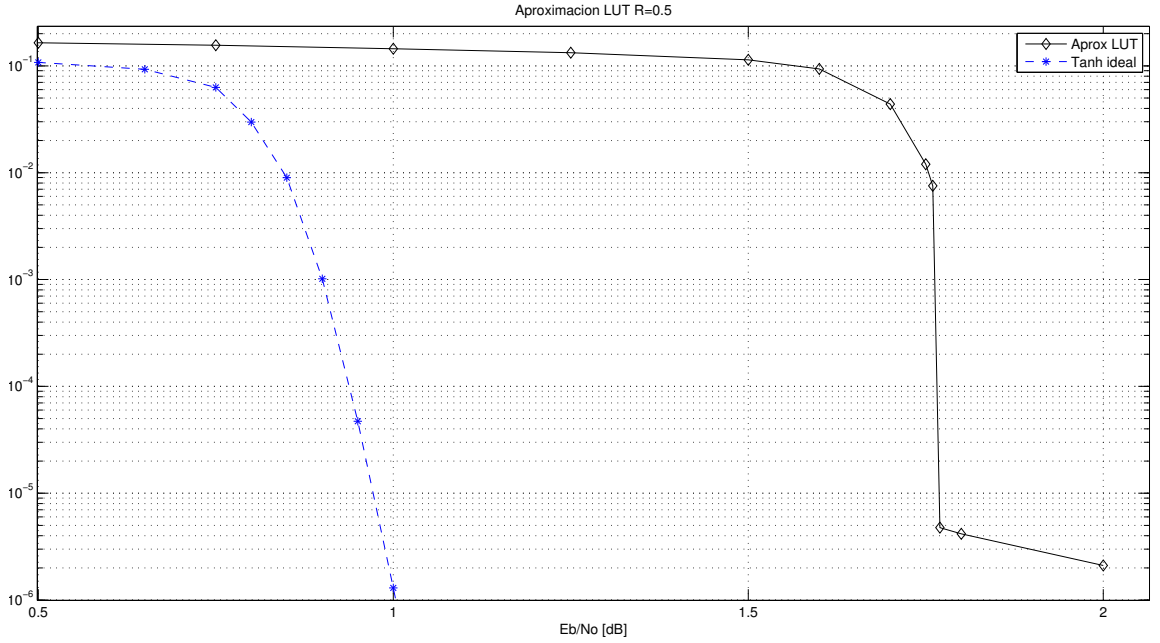

Figura 3.10: Aproximación LUT R=0.5 Con la matriz de DVB-S2

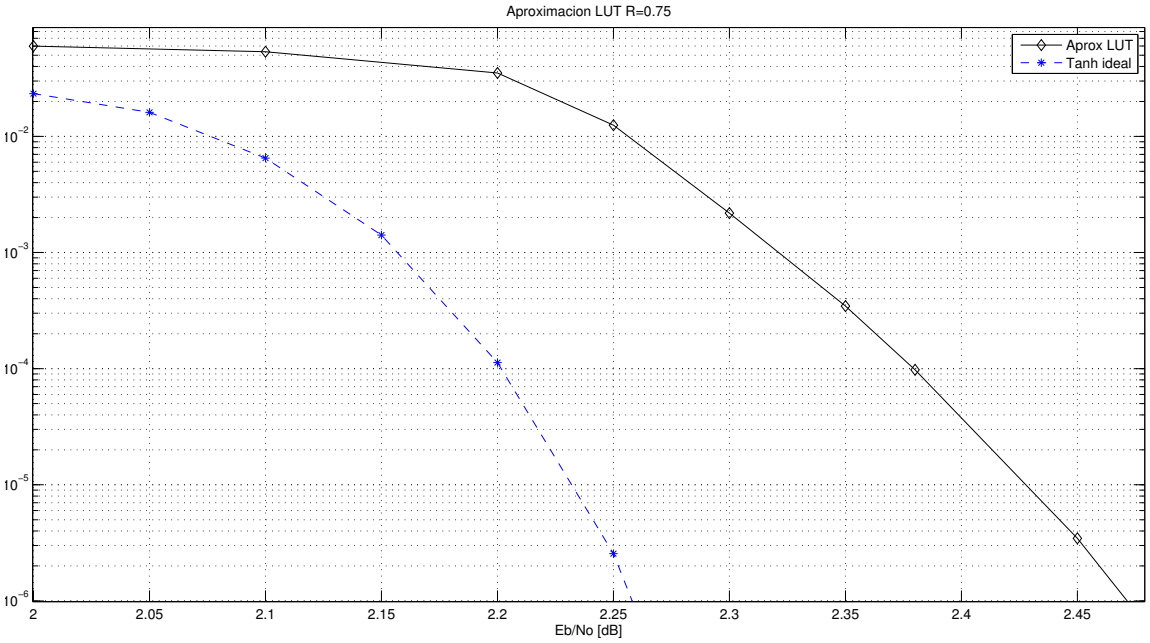

Figura 3.11: Aproximación LUT R=0.75 Con la matriz de DVB-S2

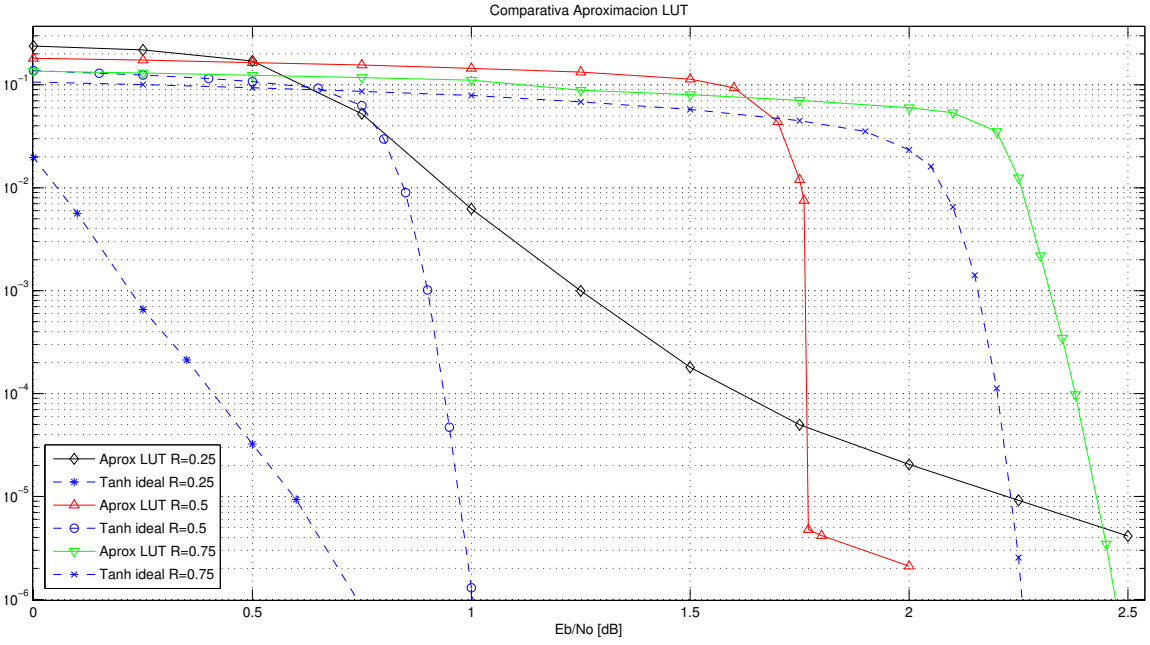

Figura 3.12: Aproximación LUT, R variable con la matriz de DVB-S2

3.2 Simplificaciones del enfoque de Gallager

Los siguientes algoritmos se basan en la simplificación de la ecuación 2.24 de distintas formas. El algoritmo minsum sustituye la ecuación $f(x)$ basándose en su monotonía, mientras que la otra solución plantea una expresión matemática alternativa para facilitar su implementación hardware.

3.2.1 Simplificación Minsum

La aproximación minsum se basa en el siguiente lema:

Lema 2. *La magnitud de $L(U \oplus V)$ es menor o igual que el mínimo de las magnitudes $L(U)$ o $L(V)$. Lo denotaremos por $|L(U \oplus V)| \leq \text{mín}\{|L(U)|, |L(V)|\}$.*

Prueba. Usando 2.24 obtenemos $|L(U \oplus V)| = f(f(|L(U)|) + f(|L(V)|)) \leq f(f(\text{mín}(|L(U)|, |L(V)|)))$. La desigualdad se sigue del hecho de que la $f(x)$ usada en 2.24 es monótonamente decreciente. Además, como $f(f(x)) = x$, tenemos $|L(U \oplus V)| \leq \text{mín}\{|L(U)|, |L(V)|\}$ □

Usando $|L(U \oplus V)| \approx \text{mín}\{|L(U)|, |L(V)|\}$, podemos aproximar 2.25 por

$$\tilde{L}_{m \to n}(x_n) = \left(\prod_{n' \in \mathcal{N}_{mn}} sign(Z_{n' \to m}(x_{n'})) \right) \times \left(\min_{n' \in \mathcal{N}_{mn}} |Z_{n' \to m}(x_{n'})| \right) \tag{3.1}$$

que es la actualización de los nodos de chequeo simplificada. Por lo tanto, hemos establecido que $\tilde{L}_{m \to n}$ y $L_{m \to n}$ tienen el mismo signo y la magnitud $\tilde{L}_{m \to n}$ es siempre mayor que $\tilde{L}_{m \to n}$. Podemos denominar a esta simplificación "minsum" ya que el algoritmo usará sumas en la actualización de los nodos de bit y búsqueda de mínimos en los nodos de chequeo. A esta aproximación se pueden aplicar correcciones para reducir la diferencia de las magnitudes.

■ Minsum Normalizado

La aproximación del apartado anterior puede mejorarse empleando una actualización de los nodos de chequeo que use una constante de normalización α, mayor que uno, dada por

$$\tilde{L}_{m \to n}(x_n) = \left(\prod_{n' \in \mathcal{N}_m n} sign(Z_{n' \to m}(x_{n'})) \right) \times \frac{\min_{n' \in \mathcal{N}_{mn}} |Z_{n' \to m}(x_{n'})|}{\alpha} \tag{3.2}$$

Aunque α debería variar con las diferentes SNRs y las diferentes iteraciones para conseguir la mejora óptima, es mantenida constante por simplicidad. Una buena manera de determinar el valor de α es por DE ("Density Evolution", un método numérico para analizar el rendimiento de los algoritmos de decodificación de paso de mensajes) [32][9][5].

■ Minsum con offset

En 2.26, el término $\log\left(1+e^{-|L(U)+L(V)|}\right)-\log\left(1+e^{-|L(U)-L(V)|}\right)$ puede aproximarse para obtener

$$L(U\oplus V)\approx \text{sign}(L(U))\text{sign}(L(V))(\text{mín}(|L(U)|,|L(V)|)-c) \tag{3.3}$$

Un método de obtención del factor de corrección c se describe en [10], donde se muestra también que el uso de una c fija en todos los cálculos de $L(U\oplus V)$ provoca una insignificante pérdida de rendimiento respecto a la decodificación BP. Un enfoque similar se presenta en [4].

Un enfoque computacionalmente más eficiente que captura el efecto del término corrector aplicado a cada nodo de chequeo se obtiene de la aproximación minsum sustrayendo una constante positiva β como se sigue

$$\tilde{L}_{m\to n}(x_n)=\left(\prod_{n'\in\mathcal{N}_m n} sign(Z_{n'\to m}(x_{n'}))\right)\times \text{máx}\left\{\min_{n'\in\mathcal{N}_{mn}}|Z_{n'\to m}(x_{n'})|-\beta,0\right\} \tag{3.4}$$

Este método difiere del método de la normalización en que los mensajes LLR menores en magnitud a β se ponen a cero, por lo tanto, despreciando su contribución en el siguiente paso de actualización de los nodos de bit.

En nuestras simulaciones solo hemos contemplado la versión sin correcciones. Podemos ver los resultados obtenidos en las figuras 3.13, 3.14, 3.15 y 3.16.

3.2.2 Simplificación por sustitución de la exponencial

Esta simplificación consiste en sustituir la exponencial en la $f(x)$ de 2.24 por una función en base 2. La idea se basa en la posibilidad de computar términos exactos de potencias de dos cuando usamos el sistema binario. Partiendo de una representación alternativa de $f(x)$, la nueva función será:

$$f(x)=\log\frac{(1+e^{-|x|})}{(1-e^{-|x|})} \tag{3.5}$$

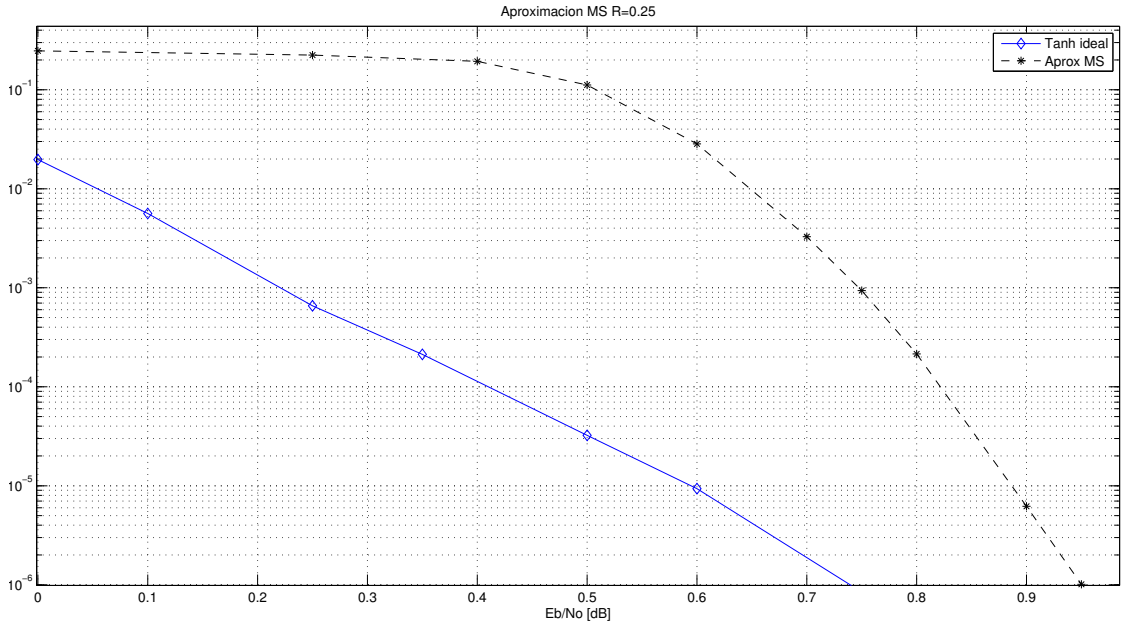

Figura 3.13: Aproximación MS R=0.25 Con la matriz de DVB-S2

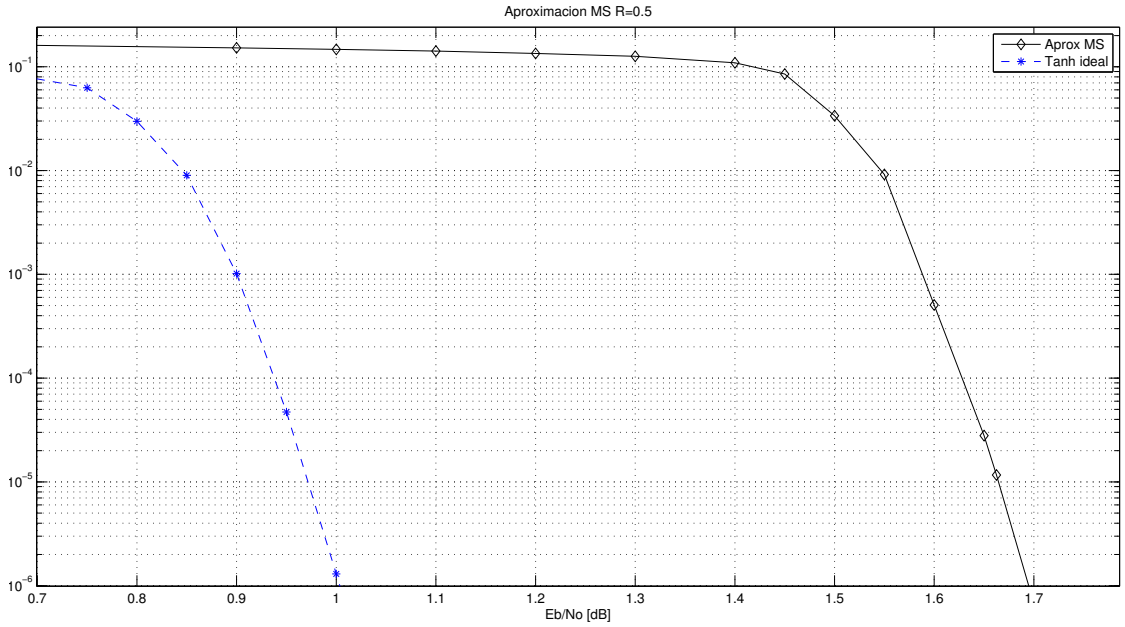

Figura 3.14: Aproximación MS R=0.5 Con la matriz de DVB-S2

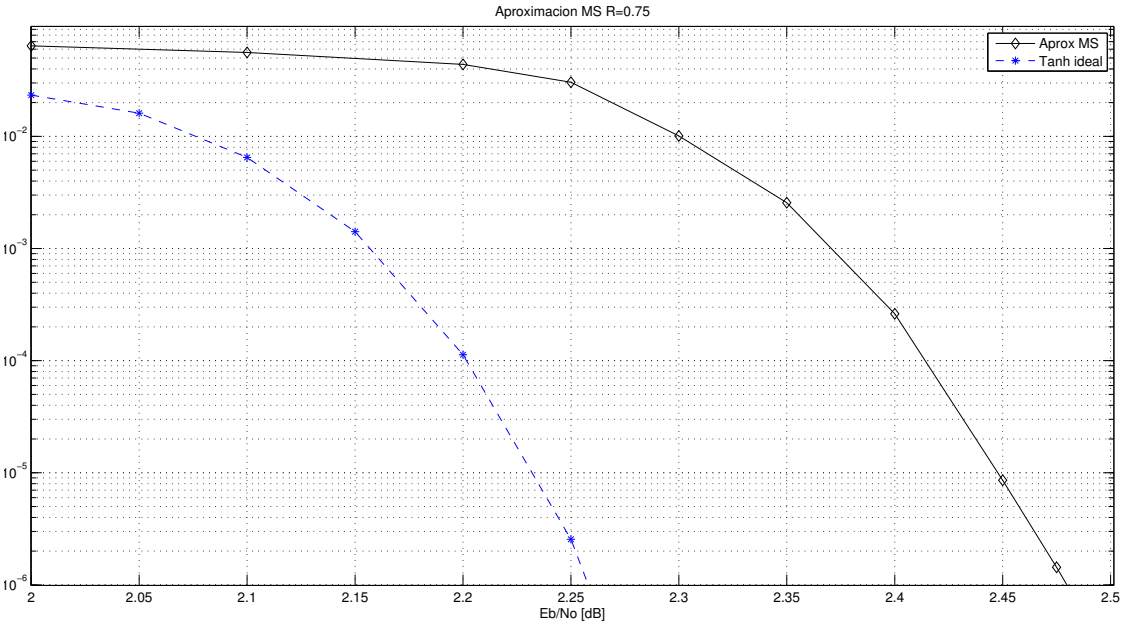

Figura 3.15: Aproximación MS R=0.75 Con la matriz de DVB-S2

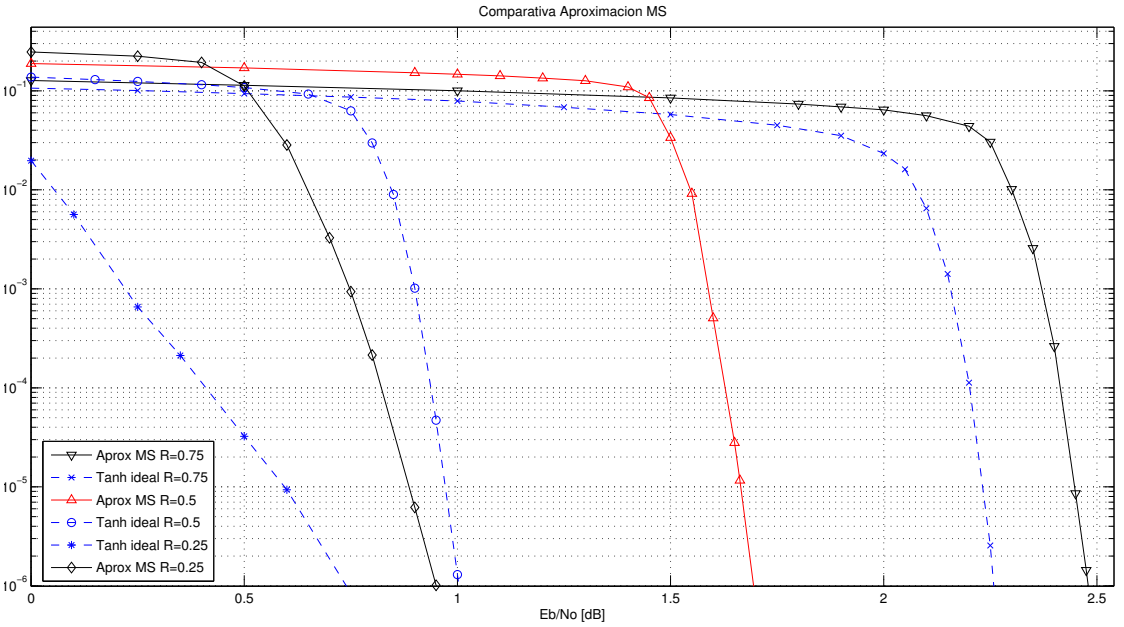

Figura 3.16: Aproximación MS, R variable con la matriz de DVB-S2

$$\Psi(x) = \log_2 \frac{(1 + 2^{-|x|})}{(1 - 2^{-|x|})} \tag{3.6}$$

La nueva definición es totalmente equivalente a la usual siempre que todas las métricas estén escaladas por el factor $\log_2(e)$, sin embargo, la potencias y logaritmos de base 2 permiten una aproximación que obtiene muy buenos resultados. La evaluación de esta nueva función puede dividirse en dos problemas más simples:

- La evaluación del término $t = 2^{-|x|}$
- La evaluación del $\log_2(1 + t)$ y del $\log_2(1 - t)$.

En cuanto al primer problema, definiremos

$$\hat{x} = |x| = x_i + x_f$$

Donde x_i es la parte entera y x_f su parte decimal. Con esto podemos reescribir t como:

$$t = 2^{-|x|} = 2^{-\hat{x}} = 2^{-x_i} 2^{-x_f}$$

Esta expresión se reduce entonces a calcular el valor de 2^{-x_f} y desplazarlo a la derecha x_i veces. Para el término 2^{-x_f} usaremos una aproximación de primer orden: $2^{-x_f} \approx 1 - x_f/2$. Esta aproximación puede calcularse fácilmente con una resta y un desplazamiento fijo a la derecha. En resumen, podemos calcular el valor de ese término con solo una resta y dos desplazamientos a la derecha.

Para evaluar de una forma simple los dos logaritmos usaremos una aproximación lineal del logaritmo [21]:

Si N es un número binario en el intervalo $2^j \leq N \leq 2^{k+1}$ con $k > j$ podemos expresarlo como:

$$N = \sum_{i=j}^{k} 2^i z_i,$$

con $z_i =$ “0” o “1”. Como $z_k = 1$ por ser el bit más significativo, podemos reexpresar N como:

$$N = 2^k \sum_{i=j}^{k-1} 2^i z_i = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i \right) = 2^k (1 + m)$$

Como $k \geq j$, m (la mantisa de N), debe estar en el intervalo $0 \leq m < 1$:

$$N = 2^k(1 + m),$$

$$\log_2 N = k + \log_2(1 + m),$$

Ahora aproximamos el $log_2(1 + m)$ por una linea recta que conecta las dos sucesivas potencias de dos. Por simplicidad se toma $a = 1$ y $b = 0$ en la ecuación $a \cdot m + b$.

$$(\log_2 N)' = k + m$$

Por ejemplo, para el número $N = 23$ ("10111") tenemos $K = 4$ y un valor de $m =$ "0,0111″. Si usamos la aproximación, $(\log_2 N)' = k + m =$ "100,0111″ = 4,43. Como puede verse, se comete un error ($\log_2(23) = 4,523561956 =$ "100, 1000011″) que puede mejorarse.

Para su implementación hardware podría usarse la solución propuesta en [2]. Básicamente, este conversor logarítmico se compone de 3 unidades funcionales: Un leading-one-detector (LOD), un registro de desplazamiento programable y un módulo de corrección del resultado. El LOD se encarga de calcular la mantisa y el registro de desplazamiento para calcular la parte decimal del resultado.

El LOD recibe como entrada el valor al que se quiere calcular el logaritmo y genera como salida una palabra con todos los bits a cero excepto en la posición del uno más significativo de la palabra de entrada. Una ROM genera la parte entera del resultado en función de la entrada activa que se usará (en ella están codificados los $\log_2$ de los enteros). Este valor será la referencia del registro de desplazamiento. Tras cada rotación se truncan los bits menos significativos (los correspondientes a la parte entera) y se genera la mantisa. Con este valor de la parte entera y de la mantisa, el error cometido es del 5.36 %, por lo que se puede emplear un circuito corrector de la mantisa que en lugar de usar una aproximación lineal entre los extremos de cada intervalo, lo divide en más subintervalos, usando para cada uno de ellos una aproximación lineal distinta, reduciendo significativamente el error.

Pueden verse los resultados obtenidos en las simulaciones en las figuras 3.17, 3.18, 3.19 y 3.20.

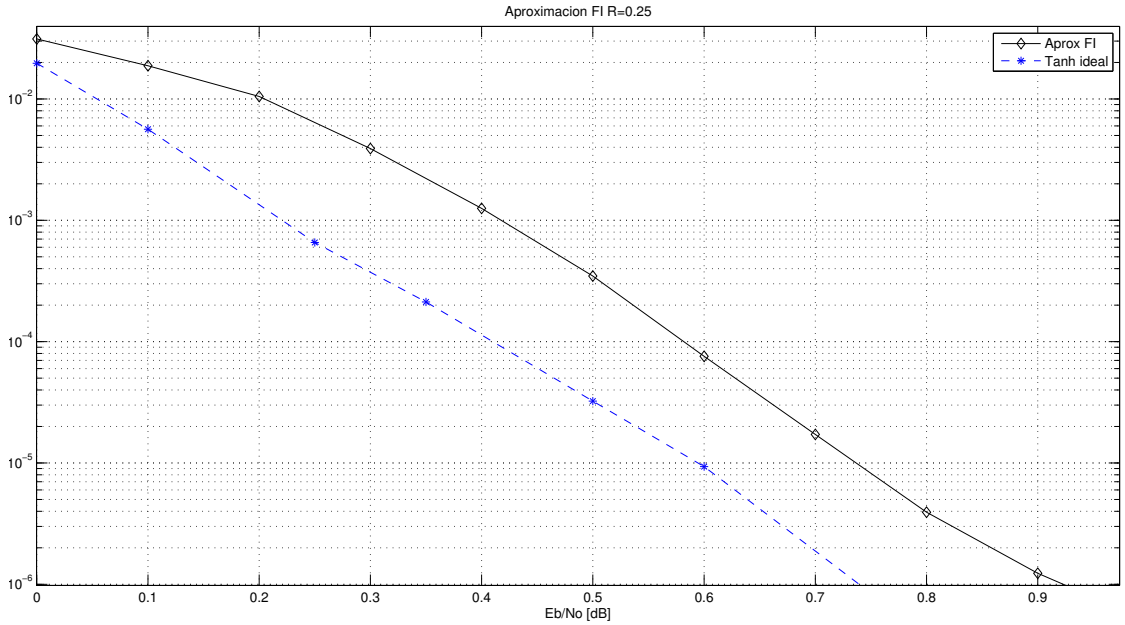

Figura 3.17: Aproximación FI R=0.25 Con la matriz de DVB-S2

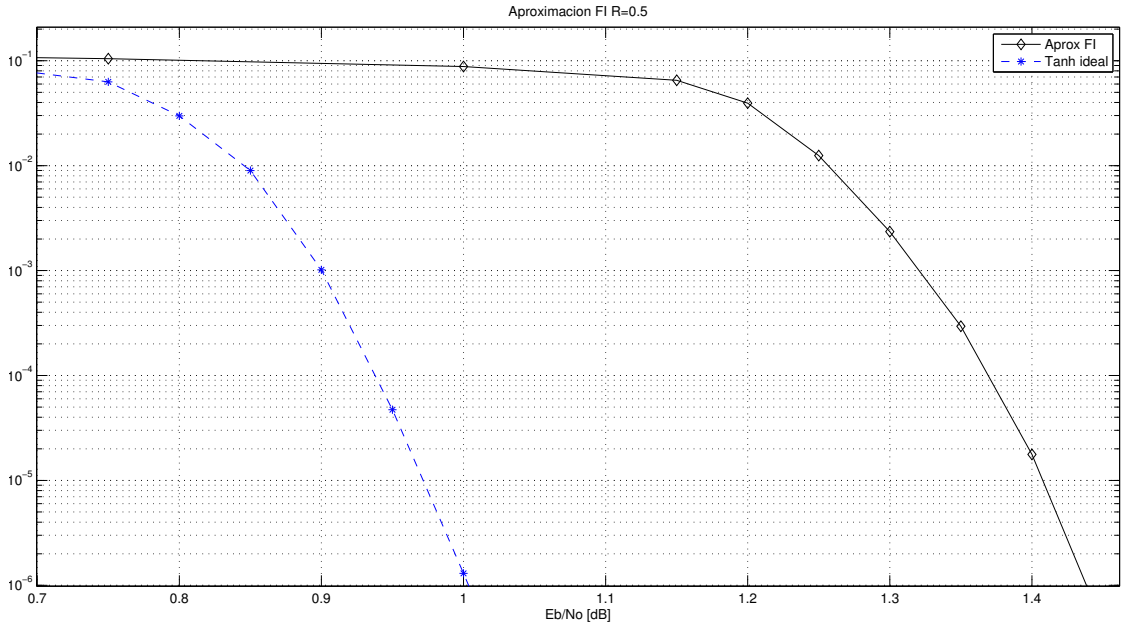

Figura 3.18: Aproximación FI R=0.5 Con la matriz de DVB-S2

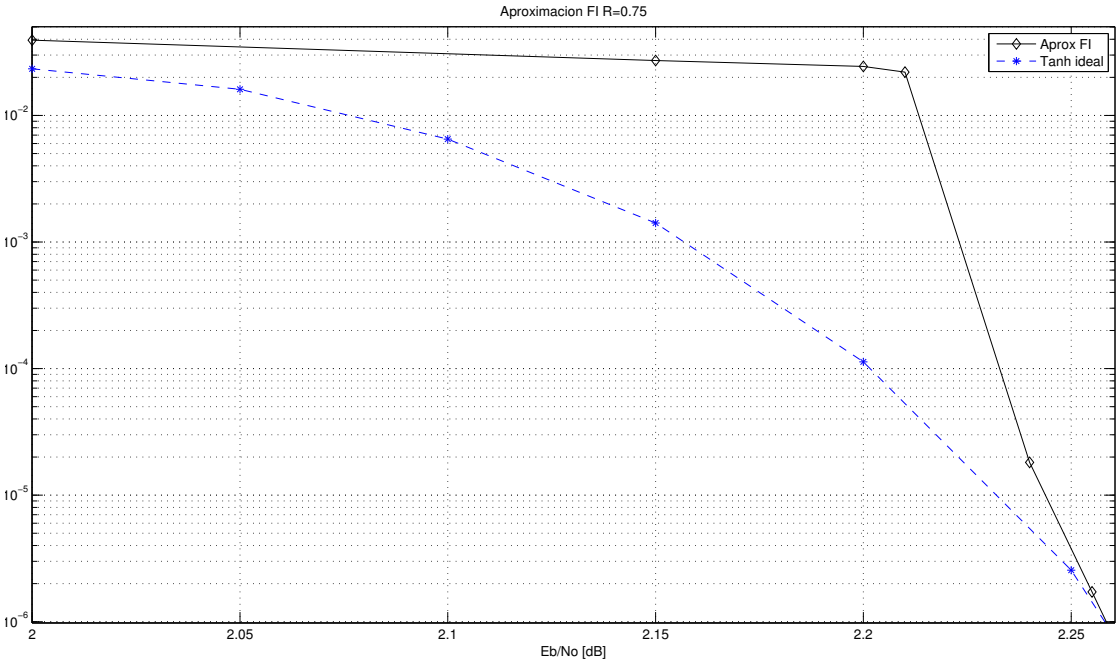

Figura 3.19: Aproximación FI R=0.75 Con la matriz de DVB-S2

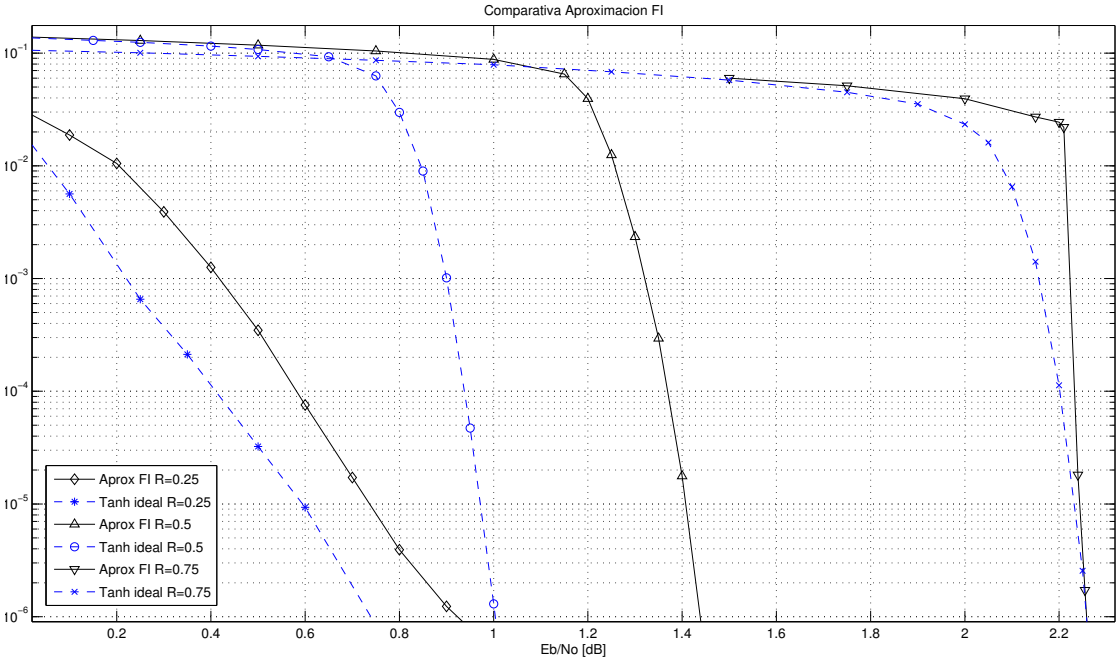

Figura 3.20: Aproximación FI, R variable con la matriz de DVB-S2

3.2.3 Comparación de los resultados

Por comodidad en la exposición de los resultados haremos las comparaciones para cada coding rate y a continuación haremos una valoración global del funcionamiento de los algoritmos. A la hora de determinar la calidad de los algoritmos solo tendremos en cuenta el ideal a la hora de ver el rendimiento de los demás, pero no lo contaremos en las valoraciones. Nos referiremos a los algoritmos como PW, LUT, FI y MS (aproximación lineal y aproximación constante de la tanh, aproximación por sustitución de la exponencial y minsum).

Para un $R = 0{,}25$, el algoritmo que mejor se comporta es el PW, seguido de cerca por el FI, el MS y finalmente el LUT, como muestra la figura 3.21. Puede observarse que la aproximación LUT se ve muy afectada por un suelo de error. En la tabla 3.5 puede verse la diferencia en la SNR para las BER 10^{-4} y 10^{-6}.

Para $R = 0{,}5$, como se ve en la figura 3.22, se sigue manteniendo que los mejores algoritmos son el PW y el FI (por ese orden), a continuación el MS y finalmente el LUT, que reduce significativamente su diferencia con el anterior. Aunque mejora el funcionamiento del algoritmo LUT, se sigue viendo afectado por un suelo de error, que esta vez se hace visible para una BER inferior a $5 \cdot 10^{-5}$. En la tabla 3.6 se resumen las distancias de los distintos algoritmos.

Finalmente, para $R = 0{,}75$, en la gráfica 3.23, los algoritmos FI y PW obtienen unos resultados muy similares y apenas se aprecian diferencias con el valor ideal para BER inferiores a 10^{-6}. El algoritmo LUT finalmente supera al MS y no se aprecia un suelo de error para esas probabilidades de error aunque es muy posible que para valores inferiores vuelva a apreciarse ese efecto. El que con este coding rate se reduzcan las diferencias entre el algoritmo ideal y las simplificaciones se debe a que se está transmitiendo mucha menos paridad, con lo que los LDPC trabajan menos y se nota muy poco su efecto. En la tabla 3.7 se resumen los datos.

En resumen, podemos observar que cuanto mayor es el coding rate, mejor se comportan los algoritmos, pero obviamente, al tener menor protección ante los errores, la BER obtenida va empeorando. El algoritmo que obtiene los mejores resultados es el PW pero es el que tiene una implementación más costosa. El siguiente algoritmo es el FI, que además es el de implementación más simple y rápida. Además, las diferencias con el PW no son demasiado acusadas, por lo que resulta ser la mejor opción. Hay que decir también, que tanto el algoritmo LUT como el PW pueden mejorarse si se toman más intervalos en las LUTs, con lo que también aumenta el coste de su implementación.

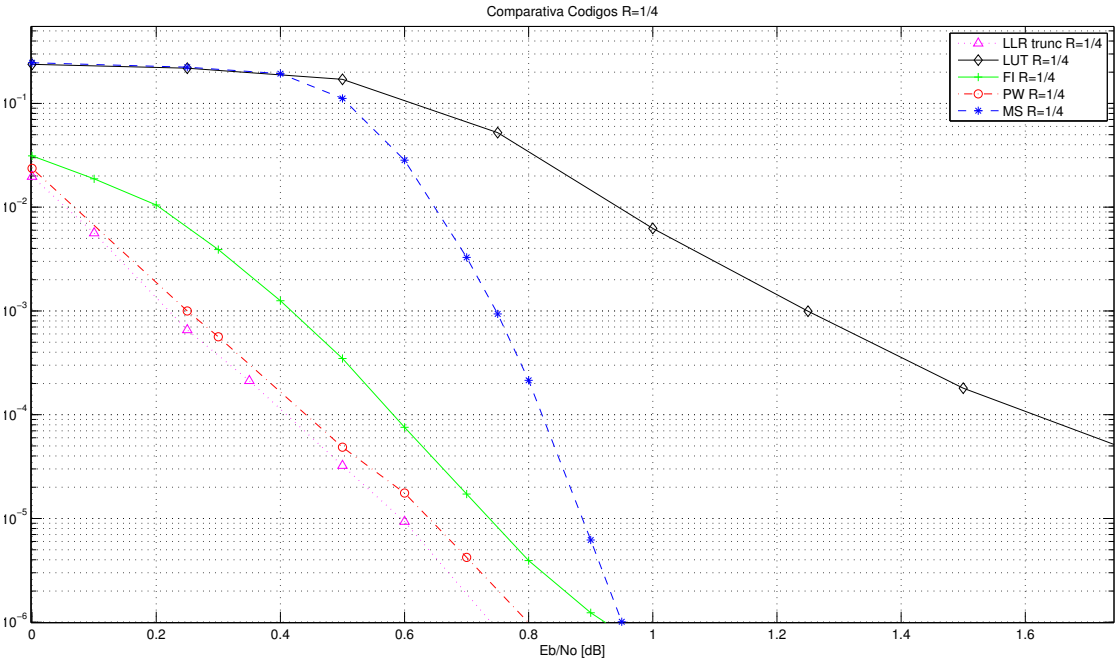

Figura 3.21: Comparativa Códigos R=0.25, N=64800bits, Nit=50

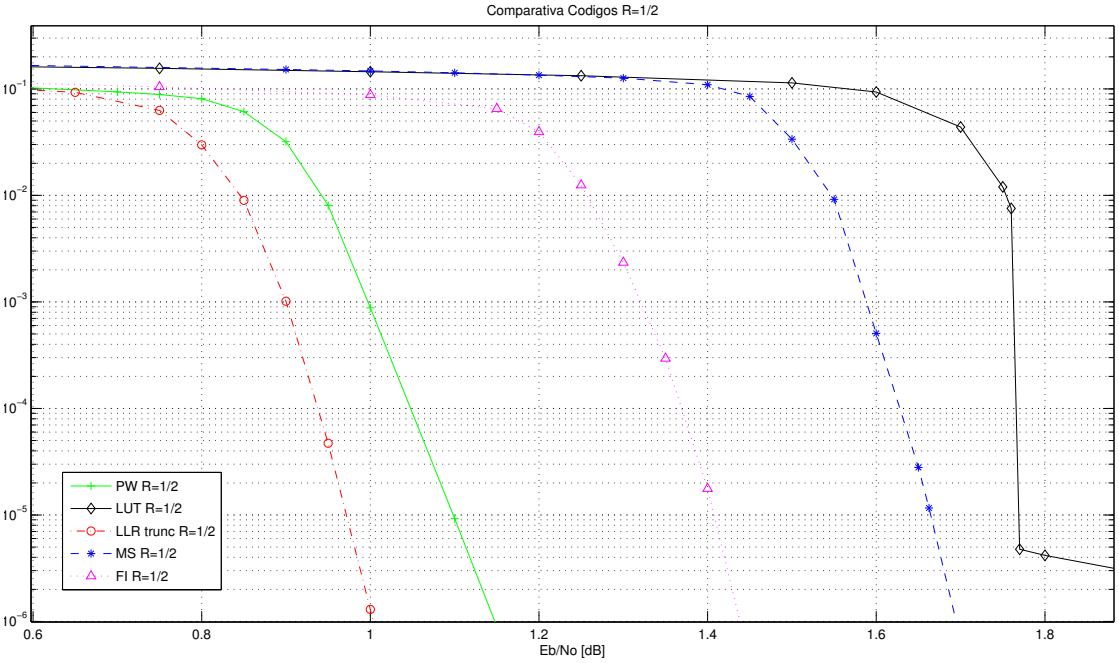

Figura 3.22: Comparativa Códigos R=0.5, N=64800bits, Nit=50

	$\text{BER} = 10^{-4}$		$\text{BER} = 10^{-6}$	
Algoritmos	SNR	Perdidas_1	SNR	Perdidas_2
LLR Ideal	0.4dB	-	0.73dB	-
Ap. PW	0.44dB	0.04dB	0.8dB	0.07dB
Ap. FI	0.58dB	0.18dB	0.93dB	0.2dB
Ap. MS	0.82dB	0.42dB	0.95dB	0.22dB
Ap. LUT	1.61dB	1.21dB	$>$ 3dB	$>$ 2dB

Tabla 3.5: Comparativa de los Algoritmos para R=0.25

	$\text{BER} = 10^{-4}$		$\text{BER} = 10^{-6}$	
Algoritmos	SNR	Perdidas_1	SNR	Perdidas_2
LLR Ideal	0.94dB	-	1dB	-
Ap. PW	1.05dB	0.11dB	1.15dB	0.15dB
Ap. FI	1.37dB	0.43dB	1.44dB	0.44dB
Ap. MS	1.63dB	0.73dB	1.7dB	0.7dB
Ap. LUT	1.77dB	0.83dB	$>$ 2dB	$>$ 1dB

Tabla 3.6: Comparativa de los Algoritmos para R=0.5

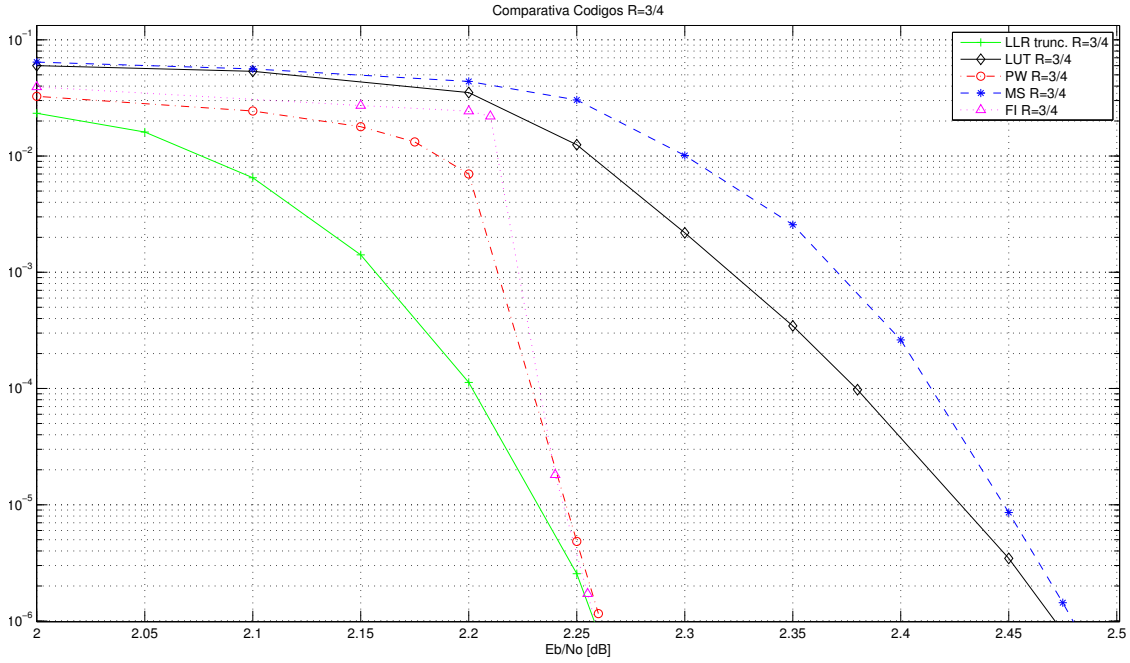

Figura 3.23: Comparativa Códigos R=0.75, N=64800bits, Nit=50

	$\text{BER} = 10^{-4}$		$\text{BER} = 10^{-6}$	
Algoritmos	SNR	Perdidas_1	SNR	Perdidas_2
LLR Ideal	2.2dB	-	2.26dB	-
Ap. PW	2.23dB	0.03dB	2.26dB	$\approx$ 0dB
Ap. FI	2.23dB	0.03dB	2.26dB	$\approx$ 0dB
Ap. LUT	2.38dB	0.18dB	2.47dB	0.21dB
Ap. MS	2.41dB	0.21dB	2.48dB	0.22dB

Tabla 3.7: Comparativa de los Algoritmos para R=0.75