FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Constellation shaping in high-speed optical fiber communications

Tiago Gonçalves Marques da Cunha

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Henrique Manuel de Castro Faria Salgado

Co-orientador: Luís Manuel de Sousa Pessoa

27 de outubro de 2021

Constellation shaping in high-speed optical fiber communications

Tiago Gonçalves Marques da Cunha

Mestrado Integrado em Engenharia Informática e Computação

27 de outubro de 2021

Abstract

The use of Probabilistic Constellation Shaping is investigated in this work combined with ASK modulation with a distribution matcher, causing a non equal symbol probability. For the Distribution Matcher a Constant Composition Distribution Matcher is used. Channel adaptation is achieved by using several error correcting codes with different code rates and by changing the symbol probability in the transmitter. In the receiver, the Belief Propagation Algorithm is used with LDPC codes from DVB-S2 with code rates (3/4,4/5,5/6,8/9,9/10). Two different channels are simulated, one dominated exclusively by white Gaussian noise and the other by a channel simulated fiber optics in MATLAB. The system is analysed at a BER $< 10^{-4}$.

Keywords: ASK, Constellation Shaping, Distribution matcher, LPDC, Probabilistic Shaping, QAM

Resumo

O uso de *Constellation Shaping* é investigado no âmbito de transmissão de dados. Um sistema ASK é combinado com um *distribution matcher* para o envio não uniforme de símbolos. Usa-se o algoritmo *Constant Composition Distribution Matcher* como *distribution matcher*. A adaptação às condições de canal é feita através da variação da *code rate* usada e no caso de ASK não uniforme, é alterada a probabilidade de envio de cada símbolo no transmissor. No receptor é usado o algoritmo de *Belief Propagation* com códigos LDPC da tecnologia DVB-S2. Os códigos usados foram (3/4,4/5,5/6,8/9,9/10) LDPC DVB-S2. O algoritmo *Probabilistic Shaping* foi simulado em ambiente MATLAB usando dois canais, um dominado por ruído Gaussiano branco e o segundo trata da transmissão em fibra óptica. O sistema é analisado para uma BER de 10^{-4}.

Keywords: ASK, Constellation Shaping, Distribution matcher, LPDC, Probabilistic Shaping, QAM

Conteúdo

Lista de Figuras

Lista de Tabelas

Abbreviations

QAM	Quadrature amplitude modulation
LDPC	Low Density Parity Check Code
DVB	Digital Video Broadcast
ASK	Amplitude shift keying
BER	Bit Error Rate
FER	Frame Error Rate
BPSK	Binary Phase Shift Keying
AWGN	Additive white Gaussian noise
SNR	Signal to Noise Ratio
MAP	Maximum a posteriori
Ml	Maximum likelihood
CCDM	Constant Composition Distribution Matcher
LLR	Log Likelihood Ratio
PS	Probabilistic Shaping
V2V	Variable to Variable
F2F	Fixed to Fixed
V2F	Variable to Fixed
PCS	Probabilistic Constellation Shaping

Capítulo 1

Introdução

1.1 Motivação

Técnicas de *Constellation Shaping* têm como propósito aumentar a taxa de transmissão de dados aproximando-se do limite teórico de Shannon. Técnicas de *Constellation Shaping* variam desde *Geometric Shaping* [20], *Probabilistic Shaping* [6], ou ainda na combinação destas duas técnicas também designado por *Hybrid Probabilistic Shaping* [7]. Este trabalho foca-se unicamente na técnica de *Probabilistic Shaping*.

Probabilistic Shaping (PS) tem vindo a ganhar relevância ao usar QAM com probabilidades de símbolo diferentes. É sabido que, para canais dominados por ruído Gaussiano branco, uma constelação QAM quadrada deve seguir uma distribuição também gaussiana [8], ou seja, os pontos interiores são enviados com mais regularidade do que os pontos mais exteriores numa constelação, permitindo aumentar a capacidade do canal.

Além de *Probabilistic Shaping* permitir um melhor desempenho comparando com técnicas tradicionais como QAM uniforme, é possível obter uma taxa de bits de informação específica dependendo do valor de SNR em questão. A granularidade da taxa de dados pode ser arbitrariamente grande permitindo um maior ou menor desempenho do sistema.

Neste trabalho foram combinados códigos LDPC da tecnologia DVB-S2 com ASK não uniforme (*Probabilistic Shaping*) e ASK uniforme, permitindo uma maior tolerância a erros em ASK uniforme e ASK não uniforme.

1.2 Objectivos

Neste trabalho pretende-se avaliar o ganho de desempenho da técnica de *Constellation Shaping* em sistemas de comunicação de alto-débito, procurando atingir a capacidade máxima do canal, tal como definido pelo limite de Shannon. Este estudo foi feito usando a modulação ASK como base de comparação.

1.3 Constribuições

As principais contribuições do presente trabalho foram as seguintes:

- maximização da capacidade do canal de um sistema de transmissão através da técnica de *Constellation Shaping*.
- Implementação de um algoritmo de *Distribution Matcher*.
- Implementação de códigos LPDC em combinação com *Constellation Shaping*.
- Simulação de um sistema ASK com probabilidade de símbolos uniforme e não uniforme em dois canais distintos: Gaussiano e fibra óptica.

1.4 Estrutura

Este trabalho é composto por sete capítulos diferentes. O primeiro é dedicado à motivação por detrás do estudo deste tema; o segundo é dedicado ao estudo de conceitos que facilitam a transição para o trabalho realizado; o terceiro capítulo aborda o estudo de códigos LDPC, principalmente códigos usados em DVB-S2 [1]; o quarto capítulo envolve o estudo de maximização da capacidade de canal em PS através do cálculo da capacidade de canal; o quinto capítulo apresenta como o sistema foi organizado, qual o algoritmo para *Distribution Matcher* que foi usado e como foi implementado; o sexto capítulo aborda os resultados obtidos por simulação e o último capítulo apresenta as conclusões obtidas.

Capítulo 2

Teria de informação e comunicação

Este capítulo tem como objectivo introduzir termos e conceitos fundamentais para o desenvolvimento deste trabalho. Serão discutidos temas relacionados com modulação QAM, conceitos de teoria de informação e de códigos corretores de erros nomeadamente códigos de blocos lineares. Na secção seguinte será discutido QAM embora neste trabalho tenha sido usado ASK nas simulações realizadas. A razão desta escolha parte do princípio que conceitos de ASK são directamente aplicáveis em QAM, visto que QAM pode também ser visto como dois sistemas ASK ortogonais. Visto que QAM é mais abrangente que ASK, escolheu-se QAM como focos de estudo na seguinte secção.

2.1 QAM

Quadrature amplitude modulation (QAM) é um tipo de modulação digital. QAM é composto pela transmissão de duas componentes ortogonais, parte real e imaginária, desfasadas de 90º entre ambas. QAM pode ser representado por

$$x(t) = v_c(t)\cos(2\pi f_c t) + v_s(t)\sin(2\pi f_c t) \tag{2.1}$$

onde

$$v_c(t) = \sum_{n=-\infty}^{\infty} a_{nc} g_T(t - nT) \tag{2.2}$$

$$v_s(t) = \sum_{n=-\infty}^{\infty} a_{ns} g_T(t - nT) \tag{2.3}$$

e a_{nc}, a_{ns} são as sequências das amplitudes transportadas nas duas portadoras; $g_T(t)$ é a forma dos impulsos e T o intervalo de símbolo.

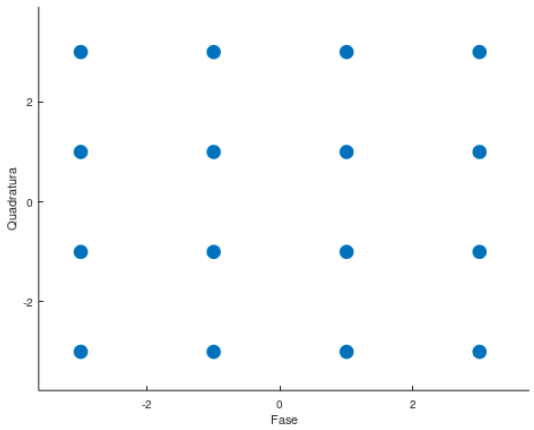

Figura 2.1: Constelação 16-QAM

QAM é gerado ao separar uma série de bits em duas séries de bits diferentes, por um conversor série-paralelo. Estas séries irão ser convertidas para valores analógicos sendo multiplicadas por uma de duas portadoras, sendo que as duas portadoras estão rodadas de 90 graus entre si. O sinal é somado e posto na saída. Um esquemático possível está representado na figura 2.2 e a respectiva constelação na figura 2.1.

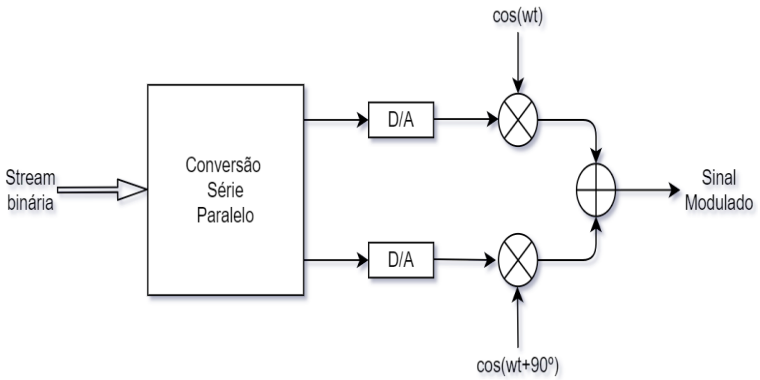

Figura 2.2: Geração sinais QAM

Um sinal QAM composto por quatro pontos, produzirá um sinal como o da figura 2.3. Neste caso, as transições apenas envolvem variações de fase, porém em geral também apresenta variações em amplitude.

A ordem de uma constelação depende do número de pontos. O número de bits necessários para codificar um símbolo é igual a $N = \log_2(M)$, sendo N o número de bits por símbolo e M, o número de pontos total da constelação . No caso da figura 2.1, o número de bits necessários para codificar um símbolo é igual a $\log_2(16) = 4$ bits/símbolo.

O equivalente banda-base de um sinal QAM pode ser representado na forma:

$$x_{bb}(t) = \sum_{n=-\infty}^{\infty} a_n g_T(t - nT) \tag{2.4}$$

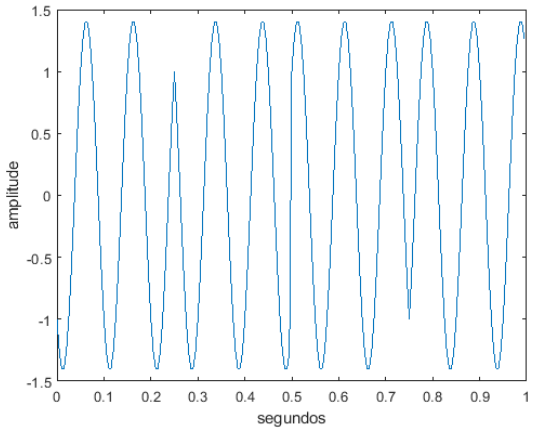

Figura 2.3: QAM no domínio dos tempos

onde a_n é a sequência de valores seleccionados que definem a constelação de sinal, correspondentes aos símbolos de informação e $g_T(t)$ é a forma dos impulsos.

A densidade espectral de potência do sinal, $S_x(f)$, é obtida tomando a transformada de Fourier da autocorrelação do processo aleatório $x_{bb}(t)$. Se $x_{bb}(t)$ tiver média nula então é possível mostrar que [23]

$$S_x(f) \propto |G_T(f)|^2 \tag{2.5}$$

onde $G_T(f)$ é a transformada de Fourier de $g_T(t)$. Considerando impulsos rectangulares conforme representado na figura 2.4, então $G_T(f) = AT\text{sinc}(fT)e^{-j\pi fT}$

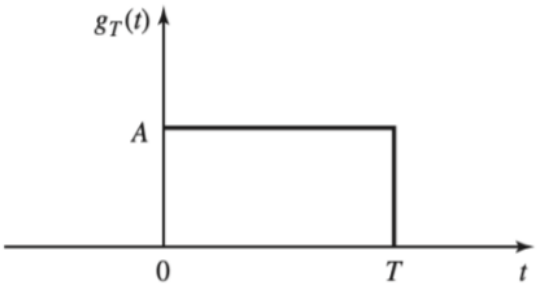

Figura 2.4: Impulso rectangular com amplitude A [23]

e

$$|G_T(f)|^2 = (AT)^2\text{sinc}^2(fT) \tag{2.6}$$

Este especto está representado na figura 2.5.

2.1.1 Filtro de cosseno elevado

Filtro de cosseno elevado é uma técnica usada para minimizar a interferência intersimbólica em canais limitados em largura de banda [4]. O espectro de onda [27] produzido por este filtro é

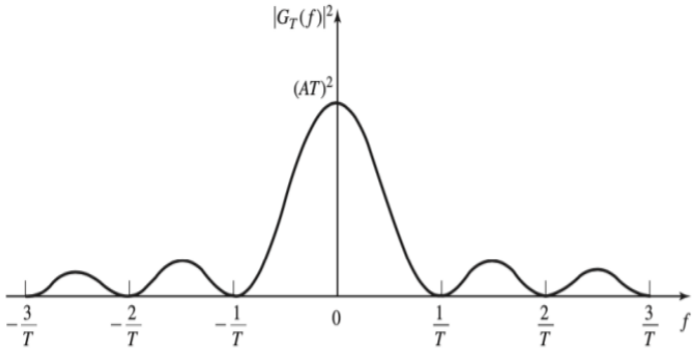

Figura 2.5: Representação do espectro do sinal QAM bandabase [23]

igual a:

$$Z(f) = \begin{cases} T_s, & 0 \leq |f| \leq \frac{1-\beta}{2T_s} \\ T_s\{1+\cos\left[\frac{\pi T_s}{\beta}\left(|f|-\frac{1-\beta}{2T_s}\right)\right]\}, & \frac{1-\beta}{2T_s} \leq |f| \leq \frac{1+\beta}{2T_s} \\ 0, & |f| > \frac{1+\beta}{2T_s} \end{cases}$$

onde β representa o factor de *roll-off* que varia entre 0 e 1. Este factor determina a banda usada pelo sistema como se pode ver pela expressão:

$$BW = \frac{1+\beta}{T_s} = (1+\beta)R_s, \tag{2.7}$$

sendo R_s a taxa de símbolos.

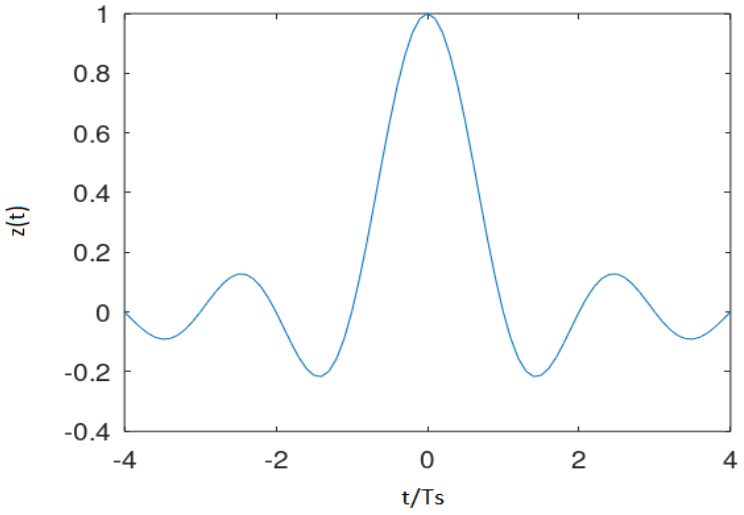

Figura 2.6: Resposta no domínio dos tempos de um filtro de cosseno elevado com $\beta = 0.2$

A resposta no domínio dos tempos e impulsional de um filtro de cosseno elevado no domínio das frequências com um $\beta = 0.2$, em banda base, estão representadas nas figuras 2.6 e 2.7, respectivamente.

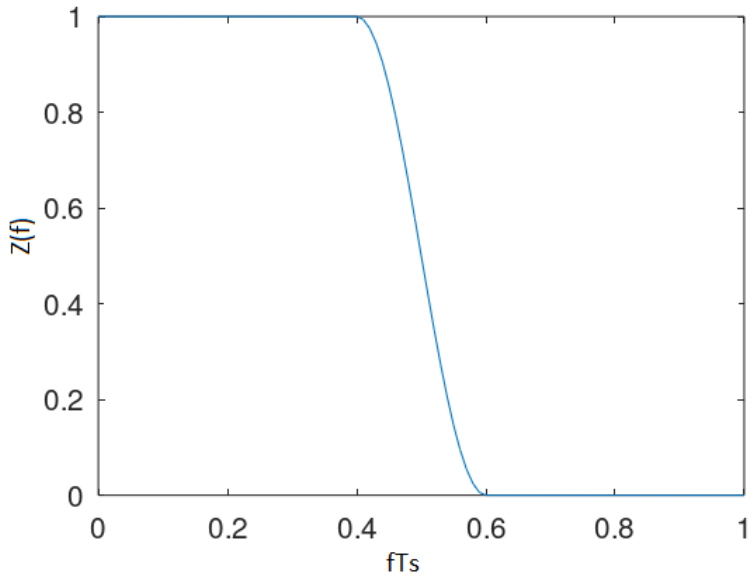

Figura 2.7: Resposta impulsional no domínio das frequências de um filtro de cosseno elevado com $\beta = 0.2$.

Sendo $H_{Fourier}(f)$ a transformada de Fourier de um sinal $h(t)$ periódico, para não haver interferência inter-simbólica, o critério de Nyquist tem que ser respeitado, ou seja:

$$\frac{1}{T_s}\sum_{k=-\infty}^{+\infty} H_{Fourier}\left(f-\frac{k}{T_s}\right)=1 \tag{2.8}$$

Para verificar que o impulso da figura 2.7 não sofre de interferência intersimbólica, veja-se a figura 2.8.

Ao fazer-se o somatório de todos estes impulsos, obtém-se um valor constante, provando que não existe interferência inter-simbólica.

2.1.2 BER

Bit error rate (BER), é um dos parâmetros que permite avaliar a qualidade de um sinal por um canal. Definida através da divisão do número de bits errados recebidos, pelo número total de bits enviados, BER é um dos factores decisivos na escolha do tipo de constelação a usar.

Usando uma constelação BPSK (símbolos -1 e +1), como representada na figura 2.9 e sendo a probabilidade de bit igual à probabilidade de símbolo errado, a *bit error rate* será então dada pela expressão 2.9, onde se admite que as probabilidades de ocorrência dos símbolos são iguais e, portanto, igual a 1/2.

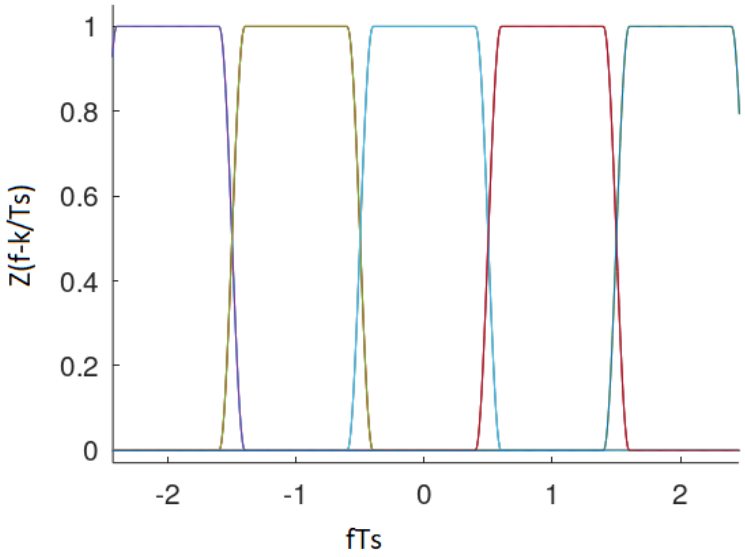

Figura 2.8: Resposta impulsional nas frequências de vários impulsos com $\beta = 0.2$.

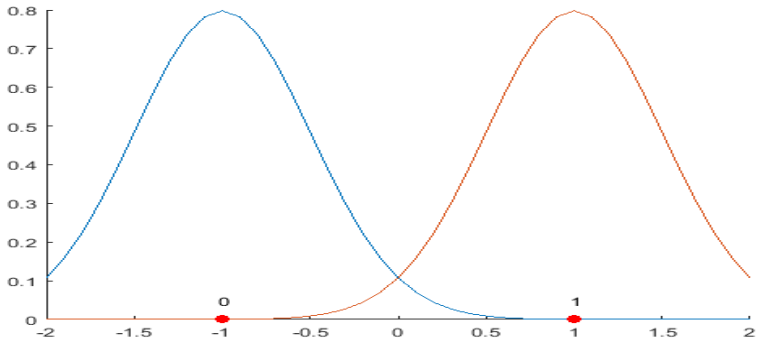

Figura 2.9: Função densidade probabilidade condicionada para um sinal BPSK

$$P_{\text{error}} = \frac{1}{2}P(S=+1|S=-1) + \frac{1}{2}P(S=-1|S=+1); \tag{2.9}$$

2.1.3 AWGN

AWGN (*additive white Gaussian noise*), é um tipo de ruído caracterizado por banda plana no domínio das frequências, caracterizado por uma distribuição normal. Este é o tipo de ruído considerado neste trabalho, devido à facilidade de cálculo envolvida. Porém, este ruído modela fracamente obstruções, efeitos de *multipath*, etc. A densidade espectral do ruído branco é constante como representado na imagem 2.10.

A potência do ruído Gaussiano branco será igual a:

$$P_{\text{AWGN}} = N_0 \cdot B \tag{2.10}$$

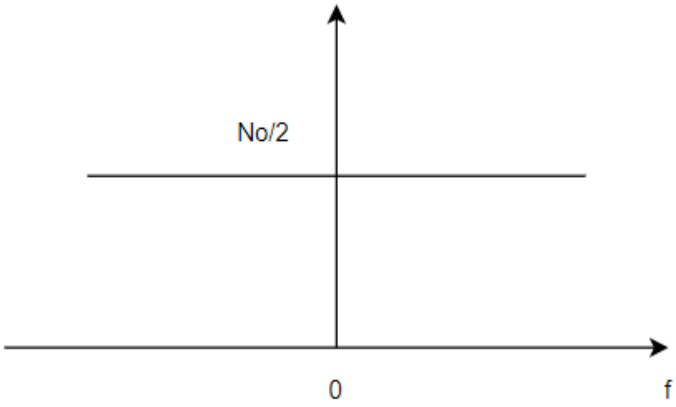

Figura 2.10: Densidade espectral de ruído gaussiano branco

Sendo N_0 a densidade espectral de ruído, em W/Hz, e B a banda considerada. Este ruído, sendo descrito a partir de um distribuição normal, pode ser expresso matematicamente, pela função densidade de probabilidade, mostrada na expressão 2.11.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.11}$$

De notar que σ^2 denota o desvio padrão ao quadrado (variância), μ, o valor central ou a média da distribuição Gaussiana.

Para a expressão 2.9, sendo o nível de decisão no valor 0 W, assumindo uma probabilidade de símbolos equiprováveis, a probabilidade de o símbolo '0' ser incorrectamente recebido com o valor '1', para certo valor de ruído, será igual a:

$$\int_0^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-(-1))^2}{2\sigma^2}}, dx \tag{2.12}$$

Por sua vez, a probabilidade de ser enviado o símbolo '1', e este ser recebido como um '0', será igual a:

$$\int_{-\infty}^{0} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-(+1))^2}{2\sigma^2}}, dx \tag{2.13}$$

A probabilidade de erro será igual à soma das expressões 2.12 e 2.13, pesada pelas probabilidades de ocorrência de cada um dos símbolos. Assume-se aqui que o nível de decisão é 0, a meio do intervalo dos símbolos 1 e -1.

2.1.4 Codificação

Embora a codificação de QAM já tenha sido mencionada brevemente, o tipo de mapeamento usado numa dada constelação afecta o comportamento do sistema. Veja-se o caso da figura 2.11.

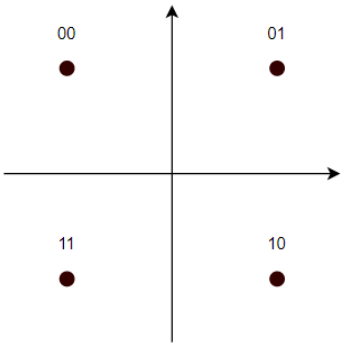

Figura 2.11: Constelação 4QAM com mapeamento natural

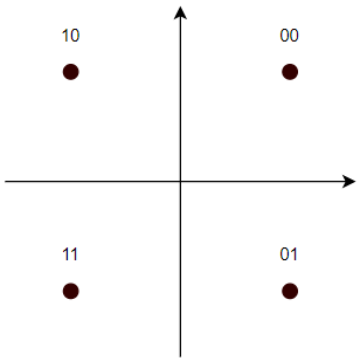

Figura 2.12: Constelação 4QAM com mapeamento de Gray

A probabilidade do ruído causar uma recepção errónea do símbolo transmitido aumenta com o potência do ruído, e para valores de E_b/N_0 não muito baixos, o mais provável é que quando um símbolo é recebido erradamente é este ter sido confundido pelo símbolo vizinho.

Da figura 2.11, vemos que o símbolo '00', quando é erradamente descodificado, será confundido mais provavelmente pelos símbolos '01' e '11', pois estes são os símbolos mais próximos. De notar que quando o símbolo enviado '00' é erradamente descodificado pelo símbolo '01', apenas um bit será erradamente recebido, ao invés do símbolo '11', que irá ter dois bits diferentes. Utilizando mapeamento de Gray, é possível garantir que dois símbolos adjacentes apenas diferem de um bit. Ao aplicar mapeamento de Gray na figura 2.11, a constelação é representada pelo diagrama de constelação da figura 2.12.

Este tipo de mapeamento pode diminuir a taxa de bits errados, para o mesmo SNR. Tal comportamento está representado na figura 2.13. Por este motivo é que será usado apenas mapeamento de Gray neste trabalho.

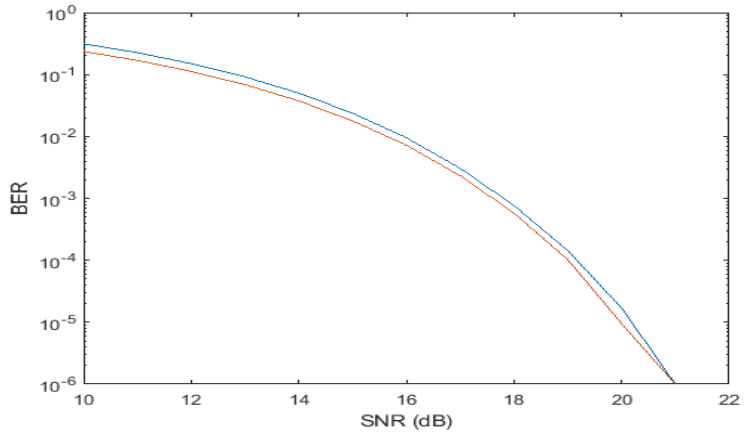

Figura 2.13: Mapeamento de Gray a laranja e mapeamento natural a azul

2.1.5 Recepção

A recepção de um sinal QAM é feito multiplicando o sinal de recebido por duas ondas sinusoidais rodadas de 90 graus entre si, assumindo que o sinal é recebido com recepção coerente, no qual as fases das portadoras estão sincronizadas com a onda recebida.

Quando se trata de recepção, existem duas maneiras de se fazer a descodificação: *hard decoding* ou *soft decoding*.

Em *hard decoding*, o sinal recebido irá ser mapeado para uma série de 0's ou 1's dependendo das distâncias entre os valores recebidos e os símbolos mais próximos, enquanto que em *soft decoding*, irão ser usados valores uma gama de valores muito maiores. Estes valores indicam uma confiança no bit recebido. Tal é o caso do algoritmo que será usado para descodificar códigos LDPC no capítulo 3. Devido a uma gama de valores maiores, *soft decoding* consegue oferecer um desempenho igual ou superior a *hard decoding*, à custa de maior complexidade [16].

Um método de geração possível de um sistema ASK com mapeamento de Gray, usado neste trabalho, passa por começar com um vector com dois elementos: (0,1). Com este vector apenas é possível gerar um sistema BPSK. Para gerar um sistema com quatro pontos (4-ASK), usa-se um método iterativo: copiam-se todos os elementos do vector anterior duas vezes, sendo que na segunda cópia inverte-se o sentido. Este vector ficaria (0,1, 1,0). Para completar este vector, preenche-se a primeira metade dos elementos do vector com zeros, e a segundo metade com uns, ou seja, (00,01,11,10). A imagem 2.14 mostra o método interactivo.

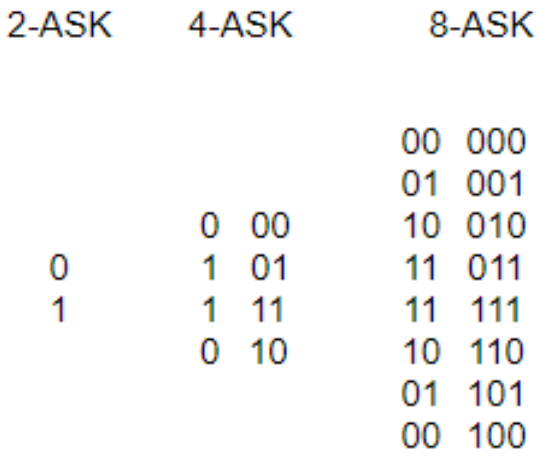

Figura 2.14: Geração de código de Gray em ASK

2.1.6 Critério de decisão de intervalos de decisao: MAP e ML

Veja-se a figura 2.15, representativa de um sinal BPSK com ruído.

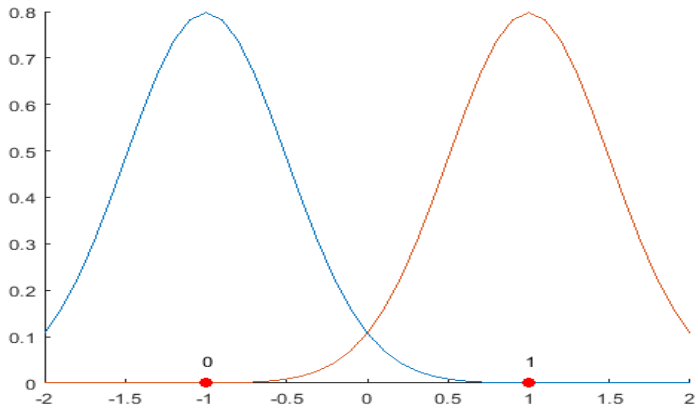

Figura 2.15: Representação da função densidade probabilidade condicionada: o símbolo com ponto da constelação -1 codifica o bit 0 e o ponto da constelação +1 codifica o bit 1.

Segundo o critério ML- *maximum likelihood*, o valor recebido 'z' irá ser descodificado para o bit B_{RX} igual a '1' ou '0' de acordo com:

$$\begin{aligned} P(z|S=-1) > P(z|S=+1) \quad B_{\mathrm{RX}}=0 \\ P(z|S=+1) > P(z|S=-1) \quad B_{\mathrm{RX}}=1 \end{aligned} \tag{2.14}$$

O critério MAP- *Maximum a posteriori* necessita de informação sobre a constelação no sentido de fazer uma decisão. Segundo o critério MAP, um valor recebido z, será descodificado para o bit B_{RX} segundo:

$$\begin{aligned} P(S=-1|z) > P(S=+1|z) \quad B_{\mathrm{RX}}=0 \\ P(S=+1|z) > P(S=-1|z) \quad B_{\mathrm{RX}}=1 \end{aligned} \tag{2.15}$$

De notar que o critério MAP precisa da distribuição dos símbolos usadas. Expandindo a expressão 2.15 com o teorema de Bayes, obtém-se:

$$P(S=x|z) = \frac{P(z|S=x)\cdot P(S=x)}{P(z)} \tag{2.16}$$

A expressão $P(z|S=x)$ segue a fórmula de uma distribuição Gaussiana, pois neste trabalho apenas é considerado AWGN. Sendo z o valor recebido, S o símbolo pertencente ao conjunto dos símbolos usados e $\mu_{S=x}$ o valor do ponto da constelação do símbolo $S=x$, a probabilidade da expressão $P(z|S=x)$ será:

$$P(z|S=x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(Z-\mu_{S=x})^2}{2\sigma^2}} \tag{2.17}$$

De notar que a expressão $P(S=x|z)$ não pode ser calculada directamente, tem que ser transformada, pelo teorema de Bayes, para uma forma do género do tipo da expressão 2.16.

Os critérios ML e MAP permitem indicar estimativas para os valores de decisão numa constelação. Numa constelação BSPK como no caso da figura 2.15, sendo A_0 e A_1 a amplitude dos

símbolos cujos pontos da constelação são -1 e +1, respectivamente, segundo o critério ML, o nível de decisão poderá ser calculado segundo as seguintes expressões, assumindo que o nível de ruído é constante em todos os pontos da constelação:

$$B_{rx} = \begin{cases} 1, & \frac{P(z|S=+1)}{P(z|S=-1)} > 1 \\ 0, & \text{Caso contrário} \end{cases} \tag{2.18}$$

Esta expressão pode ser expandida, pela expressão 2.17, para:

$$B_{rx} = \begin{cases} 1, & \frac{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-A_1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-A_0)^2}{2\sigma^2}}} > 1 \\ 0, & \text{Caso contrário} \end{cases} \tag{2.19}$$

Estas expressões podem agora ser resolvidas em ordem a z que, depois de alguma manipulação, consegue-se chegar ao limiar óptimo do critério ML, que agora define-se por γ:

$$\gamma = \frac{A_0 + A_1}{2} \tag{2.20}$$

O nível de decisão segundo o critério MAP será dada por:

$$B_{rx} \begin{cases} 0, & \frac{P(z|S=-1)\cdot P(S=-1)}{P(z)} > \frac{P(z|S=+1)\cdot P(S=+1)}{P(z)} \\ 1, & \text{caso contrário} \end{cases} \tag{2.21}$$

A expressão 2.21 pode ser simplificada para:

$$B_{rx} \begin{cases} 1, & \frac{P(z|S=+1)}{P(z|S=-1)} > \frac{P(S=-1)}{P(S=+1)} \\ 0, & \text{Caso contrário} \end{cases} \tag{2.22}$$

O valor do limite de decisão γ que definirá quando $B_{rx} = 1$ ou $B_{rx} = 0$ ocorrerá quando os termos da expressão 2.22 igualam. O processo de cálculo para se obter o limite de decisão, desta vez do critério MAP, é semelhante ao processo usado no critério ML. Depois de alguma manipulação, é possível chegar-se à expressão:

$$\gamma = \frac{\log\left(\frac{P(S=-1)}{P(S=+1)}\right) \cdot 2\sigma^2}{2 \cdot (A_1 - A_0)} + \frac{A_1^2 - A_0^2}{2 \cdot (A_1 - A_0)} \tag{2.23}$$

Por fim chega-se à seguinte expressão:

$$\gamma = \frac{\sigma^2}{A_1 - A_0} \log \frac{P(S=-1)}{P(S=+1)} + \frac{A_1 + A_0}{2} \tag{2.24}$$

A expressão 2.24 representa o valor de decisão que minimiza a probabilidade de erro entre dois símbolos adjacentes. Para um sistema M-ASK, existem $M-1$ níveis de decisão, cada nível de decisão é calculado a partir da fórmula 2.24 entre os dois símbolos mais próximos, ou seja, para

um sistema com 4 símbolos ASK ($S_1;S_2;S_3;S_4$), organizados com amplitudes crescentes no mapa de constelação, o primeiro nível de decisão é calculado a partir da expressão 2.24 entre S_1 e S_2; o segundo entre S_2 e S_3 e o último, entre S_3 e S_4. O mesmo acontece para o critério ML.

2.2 Teoria da informação

A teoria de informação permite quantificar o desempenho de um sistema de transmissão de dados dando métricas que permite quantificar o desempenho do sistema em questão. Esta secção será dedicada à introdução de conceitos necessários para este trabalho.

2.2.1 Quantidade de informação e entropia

Assumindo uma variável aleatória com uma probabilidade $P(x)$, a quantidade de informação presente [9], expressa por I, será igual a:

$$I = \log_2 \frac{1}{P(x)} \tag{2.25}$$

A base do logaritmo define a unidade de informação. Ao longo deste trabalho apenas se irá usar o logaritmo de base 2, pois a sua unidade será em bits de informação. Outras bases, como por exemplo o número de Euler, teria como unidade o 'nat'.

A entropia de uma fonte aleatória sem memória, [9], com M símbolos diferentes, cada um com uma probabilidade P_i diferente, será definida por:

$$H(X) = \sum_{i=1}^{M} P_i \cdot \log_2 \frac{1}{P_i} \tag{2.26}$$

A entropia de uma fonte pode ser vista como a quantidade de informação ou incerteza da informação produzida. Veja-se o seguinte exemplo.

A probabilidade de certo cavalo i ganhar uma corrida está presente na tabela 2.1.

Cavalo	Probabilidade de ganhar
A	1/3
B	1/3
C	1/3

Tabela 2.1: Probabilidade cavalo i ganhar

Qual é a incerteza de que um certo cavalo ganhe? Bem a incerteza terá de certeza um valor elevado, visto que como todos os cavalos têm a mesma probabilidade de ganhar, não se pode dizer muito sobre quem irá ganhar. Mas agora veja-se o seguinte caso:

Neste caso, a quantidade de incerteza sobre qual cavalo irá ganhar já será menor do que no caso anterior, pois neste cenário, o cavalo C tem um probabilidade de ganhar mais elevada, ou seja, a incerteza de quem irá ganhar a corrida é menor que no caso anterior. No primeiro cenário,

Cavalo	Probabilidade de ganhar
A	1/6
B	1/6
C	4/6

Tabela 2.2: Probabilidade cavalo *i* ganhar

a quantidade de informação, ou entropia, em bits é dada pela expressão 2.27 enquanto que no segundo cenário, a entropia é dada por 2.28

$$-(1/3\log_2 \cdot 1/3 + 1/3 \cdot \log_2 1/3 + 1/3 \cdot \log_2 1/3) \approx 1.58 \text{ bits} \tag{2.27}$$

$$-(1/6 \cdot \log_2 1/6 + 1/6 \cdot \log_2 1/6 + 4/6 \cdot \log_2 4/6) \approx 1.25 \text{ bits} \tag{2.28}$$

O conceito de entropia está bastante relacionado com a compressão de dados. Sabendo as probabilidades de cada símbolo, é possível chegar a um limite teórico de quantos bits são necessários para codificar uma mensagem, pelo menos teoricamente. Assumindo as probabilidades da tabela 2.2, cada símbolo pode, teoricamente, ser escrito com 1.25 bits. Para uma mensagem com 1000 símbolos, podia-se codificar essa mensagem com $1000 \cdot 1.25 = 1250$ bits de informação. A taxa de bits informação por segundo de uma fonte com uma entropia $H(X)$, a uma taxa de R_s símbolos por segundo é dada por:

$$R_{sistema} = R_s \cdot H(X) \tag{2.29}$$

A entropia de uma fonte com M símbolos diferentes, será sempre máxima quando os símbolos são equiprováveis. Usando apenas dois símbolos, sendo P(A)=1-P(B), o gráfico da entropia está representado na figura 2.16 em função de $P(A)$.

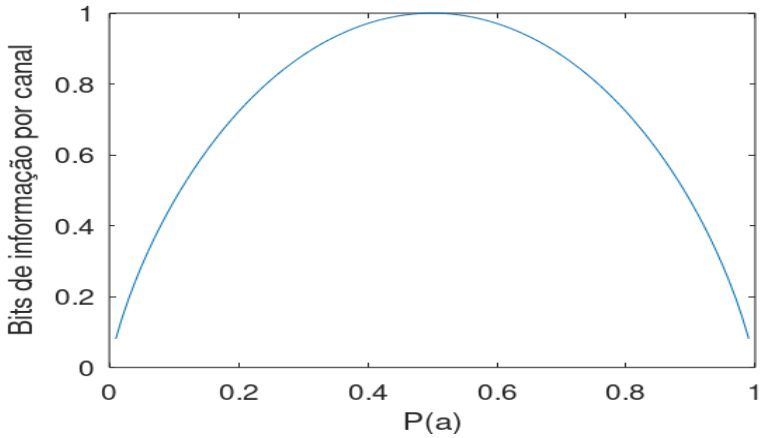

Figura 2.16: Gráfico de entropia para uma variável em função de P(A), desde zero até 1. A escala vertical, representada em bits de informação, varia desde zero a um.

2.2.2 Compressão

Compressão de dados não é um tema de verdadeira importância neste trabalho, mas um algoritmo que será discutido mais à frente, partilha muitas semelhanças com um dos algoritmos usado para compressão de dados, nomeadamente *aritmetic encoding*. Por este motivo é que foi incluído esta subsecção neste trabalho.

Aritmetic encoding pode ser visto como uma espécie de compressão de dados Huffman, sem a restrição de que as probabilidades têm que ser uma potência de 2 [14]. Este algoritmo faz parte de uma família de algoritmos de compressão de dados sem perdas.

Um esquema representativo do algoritmo de *encoding* está representado na figura 2.17, correspondente à distribuição de probabilidades dos símbolos da Tabela 2.3.

Símbolo	Probabilidade
A	1/6
B	1/6
C	4/6

Tabela 2.3: Probabilidades de símbolo i aparecer com probabilidade P_i

Aritmetic encoding permite codificar uma sequência de símbolos por bits ao atribuir intervalos a cada símbolo, desde zero até um. O tamanho do intervalo será proporcional à probabilidade do bit sendo que os intervalos não se sobreponham. Aquando da escolha de certo símbolo, dentro do seu intervalo definido, irão ser calculados novos intervalos dentro do intervalo do símbolo escolhido, repetindo-se assim o processo até que toda a sequência a codificar seja mapeada num único número. Se a mensagem a ser codificada for: (C B B), o algoritmo funcionará como indicado na figura 2.17.

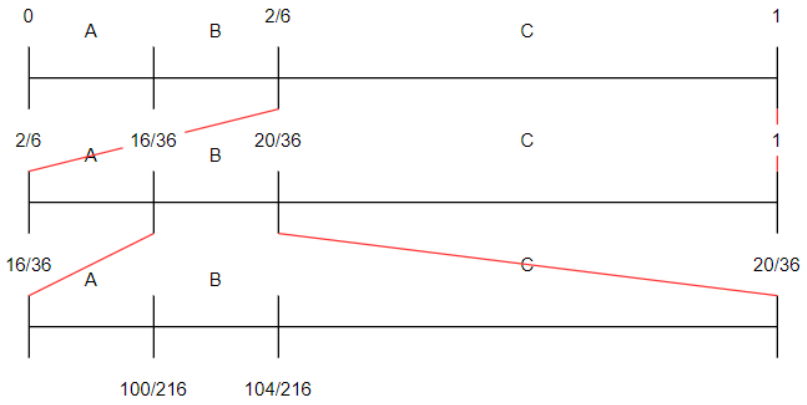

Figura 2.17: Aritmetic encoding com mensagem CBB

Desde que seja usado um valor em binário no intervalo de $[\frac{100}{216}; \frac{104}{216}]$, um descodificador seria capaz de obter a sequência original codificada. Este algoritmo será utilizado na discussão do *Distribution Matcher* no sub-capítulo 5.1.

2.2.3 Capacidade de Canal

A capacidade de canal permite identificar a taxa de transmissão de dados máxima para a qual os dados podem ser transmitidos com uma probabilidade de erro arbitrariamente pequena, ou seja, desde que $R_{sistema} \leq C$, existe um código corretor de erros que permite tornar a probabilidade de erro o mais baixa quanto o pretendido.
A capacidade de um canal é definido por:

$$C = I(X,Y) \tag{2.30}$$

Sendo $I(X,Y)$ a informação mútua entre X e Y. A informação mútua é definida por:

$$I(X,Y) = \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} P(x_j, y_k) \log_2 \left(\frac{P(y_k, x_j)}{P(y_k) \cdot P(x_j)} \right) \tag{2.31}$$

A informação mútua também pode ser expressa a partir da seguinte expressão:

$$I(X,Y) = H(X) - H(X|Y) \tag{2.32}$$

Todas as equações desta secção foram retiradas de [9].

2.2.4 Canal simétrico e binário

Um canal simétrico e binário pode ser representado por um modelo gráfico que permite caracterizar a transmissão de dados por um canal.

Um canal simétrico binário com uma probabilidade de erro p está representado na figura 2.18. De notar que nesta figura, todas as probabilidades condicionadas estão representadas. É possível então calcular o valor da capacidade do canal, como descrito na expressão 2.30.

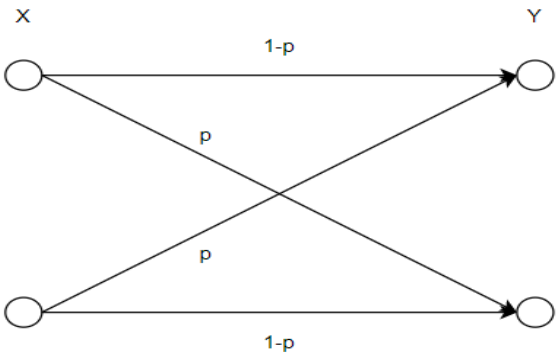

Figura 2.18: Canal simétrico binário

2.2.5 Limite de Shannon

O limite de Shannon define o limite máximo de informação no qual a comunicação sem erros é possível num canal ruidoso.

Esse limite, [9], é quantificado pela capacidade do canal definido por:

$$C = B \cdot \log_2(1 + SNR) \tag{2.33}$$

Esta é uma equação que relaciona a capacidade de um canal em bits por canal, com a banda usada e o valor do SNR.

Um canal QAM pode ser representado esquematicamente pela figura 2.19.

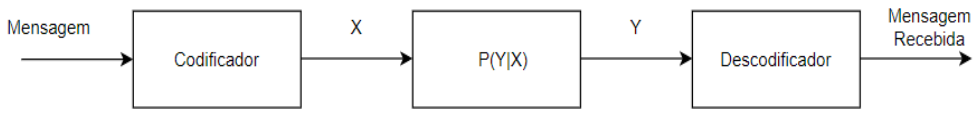

Figura 2.19: Canal ruidoso

2.3 Divergência Kullback-Leibler

Divergência Kullback-Leibler é uma 'medida' da diferença entre duas distribuições. Esta medida não é uma métrica, pois não respeita características de uma métrica tais como simetria, porém fornece uma indicação de quão diferente são duas distribuições. Sejam duas distribuições representadas pelas figuras 2.20 e 2.21.

Sendo $D_1(x)$ o elemento número x da distribuição D_1, a divergência de Kullback-Leibler, [9], é dada por:

$$D_{KL}(D_1||D_2) = \sum_{x \in X} D_1(x) \cdot \log\left(\frac{D_1(x)}{D_2(x)}\right) \tag{2.34}$$

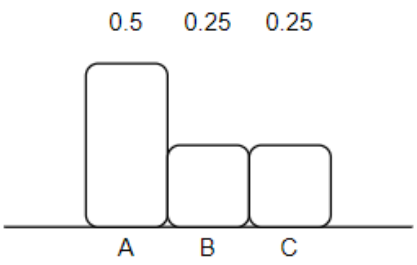

Figura 2.20: Distribuição 1

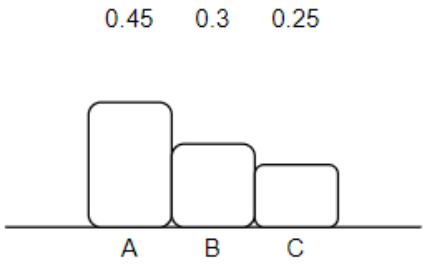

Figura 2.21: Distribuição 2

A divergência D_2 a partir de D_1, é dada por:

$$D_{KL}(D_1||D_2) = 0.5 \cdot \log\left(\frac{0.5}{0.45}\right) + 0.25 \cdot \log\left(\frac{0.25}{0.3}\right) + 0.25 \cdot \log\left(\frac{0.25}{0.25}\right) \approx 7.1 \cdot 10^{-3} \text{nats} \quad (2.35)$$

A divergência D_1 a partir de D_2 é aproximadamente $7.28 \cdot 10^{-3}$ nats, o que difere da divergência previamente calculada na expressão 2.35. Este resultado mostra a falta de simetria no cálculo da divergência, logo a divergência não representa uma métrica mas dá-nos valores relevantes aos quais se podem retirar conclusões.

2.4 Códigos corretores de erros

Códigos corretores de erros são usados para detectar e corrigir erros quando uma mensagem é transmitida através de um canal ruidoso e recebida com erros.

Códigos corretores de erros funcionam ao transmitir informação redundante extra que permitem ao receptor identificar bits errados.

Richard Hamming introduziu códigos corretores de erros, conhecidos por códigos de Hamming [13].

Em geral a correcção de erros pode ser feita de maneiras diferentes: uma delas seria retransmissão da mensagem quando o receptor identifica informação errónea na recepção; outra, que será alvo de estudo neste trabalho é enviar dados redundantes para que o receptor tenha capacidade de corrigir erros, sem necessidade de retransmissão.

Existem várias famílias de códigos de correcção de erros tais como códigos de blocos lineares, como códigos LDPC [11], códigos convolucionais, como códigos Turbo [22]. Em geral, a quantidade de erros que podem ser detectados e corrigidos numa mensagem depende da sua *code rate*, definida por:

$$R_c = \frac{k}{n} \quad (2.36)$$

A quantidade de bits da mensagem com informação é dada por k e n representa o tamanho da palavra de código. O número de bits de redundância será dado por:

$$N_{\text{bits paridade}} = n - k \quad (2.37)$$

Em geral, quanto maior for o número de bits de paridade relativamente ao tamanho da palavra de código, para o mesmo código de correcção de erros, maior o número de bits errados que podem ser corrigidos e detectados no receptor. Este comportamento será observado no capítulo dos resultados 6.

x1	x2	x3	x4	p1	p2	p3

Tabela 2.4: Exemplo de código corretor de erro (7,4): o tamanho da palavra de código corresponde a 7 bits, com os primeiros 4 bits sendo iguais à própria palavra que se quer codificar e os restantes 3 bits correspondem à informação redundante para controlo de erros (3 bits paridade)

2.4.1 Códigos lineares de blocos

Códigos lineares de blocos são códigos descritos por algum tipo de matriz, como a matriz de paridade, no qual o vector nulo é uma palavra de código válida, e a soma de duas palavras de código válidas também originam uma palavra de código válida.

Ou seja para duas palavras de código válidas: $\underline{X} = (x_1, x_2, x_3, ..., x_n)$ e $\underline{Y} = (y_1, y_2, y_3, ..., y_n)$, a palavra $\underline{Z} = (x_1 \oplus y_1, x_2 \oplus y_2, x_3 \oplus y_3, ..., x_n \oplus y_n)$ será também uma palavra de código válida.

A quantidade de erros que os códigos de blocos lineares conseguem corrigir está directamente relacionada com o conceito de distância mínima.

Distância mínima entre duas palavras de códigos corresponde à distância de Hamming mínima. Distância de Hamming corresponde à quantidade de elementos diferentes entre dois vectores válidos.

Por exemplo, sendo:

$$\begin{aligned} \underline{X} &= (100101) \\ \underline{Y} &= (101001) \\ D_{\text{Hamming}} = \underline{X} \oplus \underline{Y} &= (001100) \end{aligned} \tag{2.38}$$

Neste exemplo, a quantidade de elementos diferentes entre $\underline{X}$ e $\underline{Y}$ é igual a 2, então a distância de Hamming destes dois vectores será igual a 2. De notar que se esta distância de Hamming for a mínima entre todas as combinações de vectores, o vector $\underline{X} \oplus \underline{Y}$ continua a ser um vector válido. Então a distância de Hamming mínima será igual ao peso mínimo de todas as palavras de código, com excepção do vector nulo, sendo o peso de um vector igual à quantidade de elementos igual a 1. No exemplo acima, o peso do vector $\underline{X}$ será igual a 3.

Dependendo da distância mínima de um código, é garantido que:

$$\begin{aligned} \text{O código consegue detectar } l \text{ erros} \quad & d_{\min} = l + 1 \\ \text{O código consegue corrigir } t \text{ erros} \quad & d_{\min} = 2t + 1 \end{aligned} \tag{2.39}$$

A geração deste tipo de códigos é feita através matrizes geradoras. Neste trabalho irá assumir-se que todos os códigos lineares de blocos serão códigos sistemáticos, ou seja, os primeiros k bits corresponderão à própria mensagem que se pretende codificar, e os restantes $n-k$, os bits de paridade, estarão no fim da palavra de código, como a tabela 2.4 indica.

Dependendo da *code rate* desejada e do tamanho total da palavra de código, diferentes matrizes geradoras podem ser criadas. Uma matriz geradora é identificada como uma matriz G, que é

constituída pela matriz identidade e matriz P, senda esta última responsável por criar os bits de paridade.

A tabela 2.5, exemplifica uma possível matriz G.

1	0	0	0	1	0	1
0	1	0	0	1	1	1
0	0	1	0	1	1	0
0	0	0	1	0	1	1

Tabela 2.5: Matriz G (7,4)

Para codificar uma mensagem, basta multiplicar a mensagem que se quer codificar, com tamanho (1x4), pela matriz G, gerando uma palavra (1x7). Note-se que, como a matriz G é constituída pela matriz identidade nas primeiras quatro linhas e colunas, a palavra de código resultante terá os primeiros quatro bits iguais à própria mensagem a codificar. A partir de $\underline{M}$ que representa a mensagem que se quer codificar, a mensagem codificada será então gerada na forma de:

$$\underline{C} = \underline{M} \cdot G \tag{2.40}$$

A matriz G pode ser representada graficamente pelo chamado gráfico de Tanner, mostrado na figura 2.22. De notar que sendo o código usado (7,4), os elementos x_5, x_6 e x_7 fazem parte dos bits de paridade. Os restantes elementos são os próprios bits da mensagem a codificar.

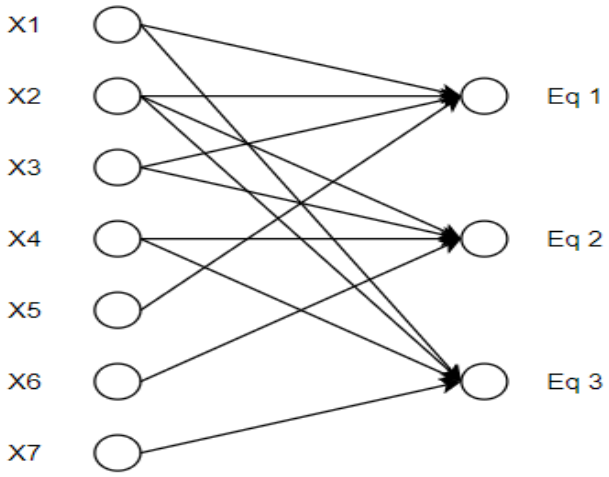

Figura 2.22: Gráfico Tanner

A matriz G também pode ser expressa a partir de equações de paridade:

$$\begin{aligned} x_1 \oplus x_2 \oplus x_3 &= x_5 \\ x_2 \oplus x_3 \oplus x_4 &= x_6 \\ x_1 \oplus x_2 \oplus x_4 &= x_7 \end{aligned} \tag{2.41}$$

Uma outra matriz de relevância é a matriz de verificação H_{ver}, [10]. Esta matriz é definida por:

$$H_{ver}^{T} = \left(\frac{P}{I_{matriz}} \right) \tag{2.42}$$

Sendo P a matriz de paridade e I_{matriz} a matriz identidade. A matriz H_{ver} gerada a partir da matriz G é descrita pela tabela 2.6.

1	0	1
1	1	1
1	1	0
0	1	1
1	0	0
0	1	0
0	0	1

Tabela 2.6: Matriz H_{ver}

A matriz H_{ver} é usada para calcular a síndrome da mensagem recebida. A síndrome (s) resulta da multiplicação da palavra de código recebida Z com H_{ver}, ou seja:

$$s = Z \cdot H_{ver}^{T} \tag{2.43}$$

Caso a síndrome origine um vector nulo, significa que a palavra recebida é uma palavra válida, caso contrário, foi recebida uma palavra com erros.

Capítulo 3

Códigos LDPC

A escolha do tipo de códigos corretores de erros terá impacto no desempenho num sistema de transmissão. Como neste trabalho, uma implementação prática eficiente em termos de *hardware* não é uma restrição importante, irão ser procurados códigos corretores que se aproximem do limite de Shannon. Dois deles são de particular interesse: códigos Turbo e LDPC.

Segundo o artigo [22], códigos Turbo funcionam melhor para baixos valores de SNR enquanto que códigos LDPC, funcionam melhor para valores mais elevados de SNR, para a mesma *code rate*. Em [6], códigos LDPC superaram códigos turbo em PS, de maneira que se optou por usar códigos LDPC.

LDPC, "*Low-density parity check-code*" é um código corrector de erros pertence à família de códigos de blocos lineares. Estes códigos são códigos com um desempenho próximo do limite de Shannon [21] que encontram muita utilidade em tecnologias como DVB-S [1]. Introduzido por Robert Gallager [11] nos anos 60, códigos LDPC foram ignorados durante anos devido às limitações de implementação de *hardware* presentes na altura. Este capítulo abordará códigos LDPC.

3.1 Matriz de paridade

A matriz de paridade $H_{paridade}$ é uma matriz de tamanho $(n-k, n)$, que expressa as equações de paridade dos códigos LDPC. Esta matriz não é a mesma representada na subsecção 2.4.1. Uma matriz de paridade $H_{paridade}$, [11], é definida por:

$$H_{paridade} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,n} \end{bmatrix}$$

Os primeiros k bits de uma palavra de um código sistemático é composto pelos bits da sequência a codificar. Os últimos $(n-k)$ bits são compostos pelos bits de paridade, como mostrado na figura 3.1.

Figura 3.1: Formato de uma palavra de código de código LDPC sistemático

A matriz de paridade $H_{paridade}$ pode ser representada pelos gráfico de Tanner. Seja uma matriz definida pela matriz 3.1, o respectivo gráfico de Tanner está representada na figura 3.2 e as equações da matriz 3.1 estão presentes em 3.2, por ordem crescente, ou seja, desde a equação 1 até à equação 3.

$$H_{paridade} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.1}$$

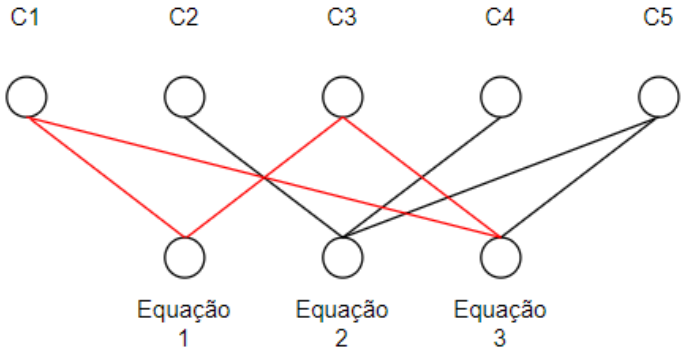

Figura 3.2: Gráfico de Tanner da matriz de paridade 3.1

.

$$\begin{aligned} c1 \oplus c3 &= 0 \\ c2 \oplus c4 \oplus c5 &= 0 \\ c1 \oplus c3 \oplus c5 &= 0 \end{aligned} \tag{3.2}$$

3.1.1 Características códigos LDPC

Códigos LDPC são caracterizados por uma matriz escassa, ou seja, com um número reduzido de 1s. Os códigos LDPC podem ser categorizados em duas classes distintas:

- Códigos regulares
- Códigos irregulares

Um código LDPC é considerado regular quando o peso de cada coluna e de cada linha da matriz $H_{paridade}$ é constante. O peso de uma linha ou de uma coluna corresponde ao número de elementos diferentes de zero nessa linha ou coluna. Um código com um número variável de elementos não nulos entre as diferentes linhas ou colunas diz-se irregular. Em geral, códigos LDPC irregulares superam códigos LDPC regulares em termos de desempenho [18]. A matriz 3.1 diz-se então irregular, pois o número de 1s em cada linha não é constante, bem como o número de 1s em cada coluna.

A *coding rate* de um código LDPC depende do número de linhas da matriz de paridade: para uma matriz com $(n-k)$ linhas linearmente independentes, existirão $(n-k)$ bits de paridade. A *coding rate* será então:

$$R_c = \frac{k}{k+(n-k)} = \frac{k}{n} \tag{3.3}$$

Um dos factores que contribui para o desempenho de um código LDPC é o tamanho de um ciclo. Um ciclo é um percurso de vértices conectados que começa e acaba no mesmo vértice, passando pelo mesmo vértice não mais que uma vez. Na figura 3.2 está demonstrado um ciclo cujo trajecto é (c1 -> Equação 3 -> C3 -> Equação 1 -> C1). A distância deste ciclo é de quatro. Este aspecto é importante pois o valor mínimo de um ciclo relaciona-se com o desempenho do código LDPC. Para se obter maior desempenho, a distância mínima de um ciclo deve ser grande pois a existência de uma distância mínima de um ciclo pequeno, como o da figura 3.2, leva a uma perda de desempenho devido a uma não convergência ou convergência lenta no algoritmo de descodificação [18].

3.1.2 Construção códigos LDPC

De acordo com Gallager [11], códigos LDPC podem ser gerados a partir de permutações de colunas de sub matrizes, como mostrado no seguinte exemplo:

Para uma matriz com $\underline{n}$= 12, $\underline{f}$= 4 e $\underline{j}$= 3, sendo $\underline{n}$ o tamanho da palavra de código, $\underline{f}$ o peso de cada linha e $\underline{j}$, o peso de cada coluna, um código LDPC pode ser gerado a partir de uma sub-matriz, como mostrado na matriz 3.4.

$$h_{paridade} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.4}$$

Através da sub-matriz de tamanho (3,12), equação 3.4, pode-se gerar uma matriz de paridade com tamanho $(\frac{\underline{n}}{\underline{f}} \cdot \underline{j}\ ,\ \underline{n})$, concatenando verticalmente $\underline{j}$ sub-matrizes de tamanho $(\frac{\underline{n}}{\underline{f}}, \underline{n})$ através de permutações das colunas da matriz 3.4. Neste caso, através da geração de duas sub-matrizes (3,12)

e concatenando-as com a sub-matriz 3.4, é possível gerar a matriz de paridade 3.5.

$$H_{paridade} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

As permutações das colunas podem ser feitas de uma maneira aleatória. Porém, a construção de matrizes de controlo de paridade não garantem que haja uma distância de ciclo mínimo maior que quatro, o que pode prejudicar o desempenho deste código LDPC.

3.2 Codificação

A codificação de códigos LDPC construídos pelo método 3.1.2 em geral não é um problema trivial. A matriz de paridade 3.5 é uma matriz que gera um código LDPC com uma palavra de código de 12 bits, sendo 3 bits de informação e 9 bits de paridade. Usando um código LDPC sistemático, as primeiras 3 colunas corresponderão aos 3 bits de informação e as restantes colunas corresponderão aos bits de paridade. Seja $\underline{M}$ uma mensagem a codificar definida por (m_1, m_2, m_3). Neste caso, a palavra de código $\underline{C}$ definida por $c_1, c_2, c_3, ..., c_{12}$, terá os três primeiros bits definidos por (m_1, m_2, m_3). Porém, os restantes nove bits $(c_4, c_5, ..., c_{12})$ terão que ser calculados. Veja-se as primeiras duas linhas da matriz de paridade 3.5, replicadas aqui por conveniência.

$$l_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

$$l_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.7}$$

A primeira linha l_1, é de fácil codificação pois como os primeiros 3 bits são conhecidos, apenas basta fazer o cálculo para o primeiro bit de paridade (c_4), como mostrado na equação 3.8.

$$c_1 \oplus c_2 \oplus c_3 \oplus c_4 = 0 \tag{3.8}$$

O bit c_4 representa o primeiro de nove bits de paridade, que é de fácil cálculo. Porém, a segunda linha da matriz 3.5, indicada na equação 3.7, é definida pela equação:

$$c_5 \oplus c_6 \oplus c_7 \oplus c_8 = 0 \tag{3.9}$$

Esta equação não pode ser resolvida imediatamente pois os bits de c_5 a c_8 são bits de paridade, que dependem da mensagem a codificar. Não existe maneira de resolver esta equação neste momento, pois estes bits não são ainda conhecidos. Estes serão calculados resolvendo um sistema de equações que envolve todas as equações da matriz de paridade. No caso da matriz 3.5, teremos nove incógnitas (que correspondem ao número de bits de paridade) para nove equações (que correspondem ao número de linhas da matriz de paridade).

O problema dá-se agora quando o objectivo for escalar a matriz $H_{paridade}$ para tamanhos de palavras de código maiores. As matrizes de paridade que se irão usar neste trabalho virão da tecnologia DVB-S2. Estas matrizes têm códigos LDPC de 64800 bits no qual, dependendo da *code rate*, poderão ter um número elevado de equações: para *code rate* de 1/2, o número de equações será igual a $1/2 \cdot 64800 = 32400$. Subitamente resolver o problema da codificação desta maneira começa a tornar-se numa impossibilidade prática.

Uma outra maneira de resolver o problema da codificação passa por aplicar a mesma abordagem que foi aplicada na subsecção 2.4.1. Esta abordagem passa por criar uma matriz G no qual a palavra de código poderia ser gerada a partir de uma simples multiplicação de matrizes:

$$\underline{C} = \underline{M} \cdot G \tag{3.10}$$

Para gerar uma matriz G a partir da matriz $H_{paridade}$, pode-se usar o algoritmo de *"Gauss-Jordan elimination"*, gerando matrizes G do formato $G = [I|K]$, tal como visto na subsecção 2.4.1. Porém a matriz G resultará numa matriz que em geral não será esparsa, o que resulta numa grande complexidade de codificação [5], não sendo também de fácil implementação prática.

A codificação de matrizes de paridade pode ser imensamente mais fácil ao construir uma matriz de paridade com uma estrutura predefinida. Observe-se a matriz $H_{paridade}$ do código LDPC DVB-S2 na figura 3.3.

Embora não seja muito legível, a imagem 3.3 pode ser decomposta em duas matrizes: matriz A de tamanho $((n-k),(k))$ e matriz B de tamanho $((n-k),(n-k))$, como mostrado na expressão 3.11.

$$H_{paridade} = [A|B] \tag{3.11}$$

A matriz B é composta pela matriz identidade no qual cada elemento não nulo nessa matriz possui um 1 no elemento imediatamente na linha inferior, como mostrado na seguinte matriz 3.12:

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{3.12}$$

De notar que a matriz B no caso da figura 3.3, que corresponde aos bits de paridade, tem tamanho (32400 , 32400), pois como a *code rate* da matriz da figura 3.3 é de 1/2 e o tamanho da

Figura 3.3: Estrutura do código LDPC de DVB-S2 para *code rate* 1/2 com tamanho 64800. A azul estão representado os elementos nulos e a amarelo os elementos não nulos (igual a 1)

.

palavra de código é de 64800 bits, o número de bits de paridade será igual a $64800/2 = 32400$ bits. Esta estrutura é útil na codificação pois significa que os bits de paridade podem ser calculados sequencialmente: olhando para a primeira equação da matriz B 3.12, o único elemento com um '1' é o elemento da primeira coluna. Visto que a matriz B é a matriz que representa os bits de paridade, e apenas existe um bit de paridade a calcular, pode-se proceder imediatamente ao cálculo do primeiro bit de paridade pois este dependerá apenas da mensagem pretendida a codificar. A segunda linha da matriz B 3.12 possui elementos não nulos na primeira e na segunda coluna, o que significa que existe uma dependência entre o primeiro e o segundo bit de paridade. Porém, visto que anteriormente já foi possível calcular o valor do primeiro bit de paridade, o segundo bit de paridade apenas dependerá da mensagem a codificar e do primeiro bit de paridade já conhecido, sendo possível descobrir o valor do segundo bit de paridade. O terceiro bit de paridade dependerá apenas dos bits da mensagem que se pretende codificar e do segundo bit de paridade que já foi descoberto. Todos os bits de paridade podem ser obtidos através da expressão 3.13. A e B são as matrizes representadas em 3.11. A_i representa a linha i da matriz A, $\underline{M}$ representa a palavra a codificar e $p(i)$ o bit i de paridade que se pretende calcular.

$$p(i) = \begin{cases} \text{mod2}\left(A_i \cdot \underline{M}^T\right) \oplus p(i-1), & i > 1 \\ \text{mod2}\left(A_i \cdot \underline{M}^T\right), & i = 1 \end{cases} \tag{3.13}$$

Sendo uma matriz de paridade esparsa, ou seja, a maioria dos elementos são nulos, não é necessário guardar todos os elementos da matriz em memória, mas apenas a posição dos elementos não nulos. Neste caso, o número de operações necessárias para codificar um código LDPC da tecnologia DVB-S2 será igual ao peso de Hamming de cada linha, j_i, multiplicado pelo número de linhas da matriz de paridade $H_{paridade}$, ou seja:

$$N_{\text{operações}} = \sum_{i=1}^{n-k} j_i \tag{3.14}$$

O peso de Hamming de cada linha da matriz $H_{paridade}$ na figura 3.3, é igual a sete, excepto a primeira linha, que tem peso igual a seis.

3.3 Descodificação

A descodificação de uma palavra de código pode ser feita em conjunto com o desmodulador ou em separado. Irá ser analisado primeiramente o caso em que o descodificador LDPC está separado do desmodulador.

3.3.1 *Bit flipping algorithm*

Bit flipping algorithm é um método de descodificação rígido que aceita apenas bits provenientes do desmodulador. O algoritmo, descrito em [11], funciona da seguinte maneira:

1. Calcular a síndrome: $s = Z \cdot H_{ver}^T$. Se o resultado for um vector nulo, acaba-se a descodificação ou acabar a descodificação caso um número de iterações máximo seja alcançado.
2. Descobrir que equações de paridade apresentam desigualdades e registar quais são os elementos que fazem parte dessas equações.
3. O elemento que está presente no maior número de equações erradas é alterado.

Seja um código de Hamming (7,4), descrito pela matriz de paridade 3.15.

$$H_{paridade} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

Suponha-se agora que uma palavra de código válida $\underline{C}$: (1 0 1 1 0 0 0), ao ser transmitida, foi recebida como: (1 $\underline{1}$ 1 1 0 0 0). Como se pode ver, houve um bit recebido errado, que está destacado a sublinhado. Segundo a matriz 3.15, a primeira equação será igual a 3.16.

$$1 \cdot 1 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 0 \oplus 0 \cdot 0 = 1 \neq 0 \tag{3.16}$$

A segunda equação será igual a 3.17

$$0 \cdot 1 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \oplus 1 \cdot 0 \oplus 0 \cdot 0 = 1 \neq 0 \tag{3.17}$$

E o mesmo se repete para a terceira equação de paridade. Neste caso, todas as equações estão erradas. Quando são detectados erros, descobrem-se quais são os bits que estão presentes nas equações erradas. Neste caso, c_1 está presente em duas equações erradas, c_2 está presente em três equações erradas e daí em diante. A tabela 3.1 mostra o número de vezes em que cada bit pertence a uma equação errada.

Bit	1	2	3	4	5	6	7
Número de ocorrências	2	3	2	2	1	1	1

Tabela 3.1: Número de vezes que o bit i está presente em equações de paridade erradas

Ao alterar o bit com mais erros, neste caso o segundo bit, de 1 para 0, fazendo o cálculo da síndrome, é possível mostrar que:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \tag{3.18}$$

Ou seja, a palavra foi descodificada, neste caso, sem erros. Poderá haver casos no qual se um número suficientemente grande de erros existir, a palavra recebida possa ser descodificada para outra palavra válida que não a originalmente enviada. O uso deste algoritmo implica decisões rígidas sobre qual o símbolo recebido. Tal como mostrado na subsecção 2.1.6, existem dois métodos de calcular os limites dos símbolos.

Sejam dois pontos de constelação: (A_{-1} e A_{+1}), com probabilidades de ocorrência: (0.7 e 0.3), respectivamente, como mostrado na figura 3.4.

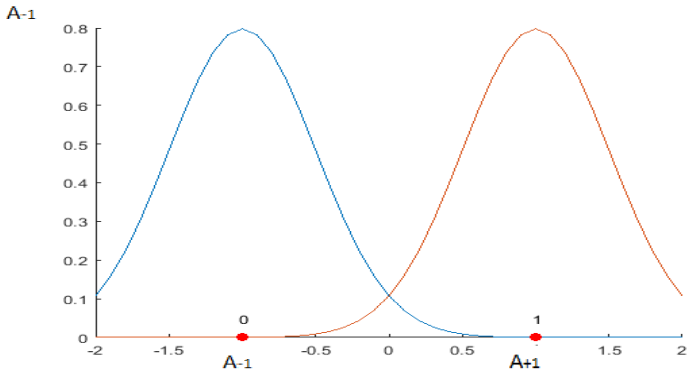

Figura 3.4: Constelação BPSK. A probabilidade do símbolo $P(S=-1)$= 0.7 e a probabilidade do símbolo $P(S=+1)$= 0.3. O símbolo A_0 corresponde ao bit 0 e A_{+1} corresponde ao bit 1

.

Segundo o critério ML, o valor de decisão está representado na equação 3.19:

$$\gamma = \frac{(-1+1)}{2} = 0 \tag{3.19}$$

Segundo o critério MAP, o valor de decisão que depende do nível de ruído, será igual a:

$$\gamma = \frac{\sigma^2}{1-(-1)} \log\left(\frac{0.7}{0.3}\right) + \frac{1+(-1)}{2} \tag{3.20}$$

A probabilidade de bit errado numa constelação BPSK é dada pela figura 3.5. Como se pode ver, o critério MAP é superior ao critério ML.

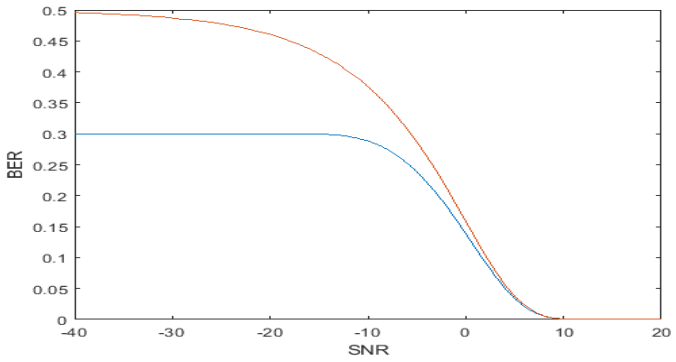

Figura 3.5: Probabilidade de erro do critério MAP a azul vs ML a laranja numa constelação BPSK na qual um dos símbolos tem probabilidade de ocorrência 0.3 e o outro 0.7. A potência da constelação é igual a 1W, e assumiu-se uma largura de banda de 1Hz.

.

Como a capacidade de correcção de erros do algoritmo *Bit-Flipping* depende de decisões rígidas no desmodulador, é de esperar que este algoritmo tenha um desempenho superior usando o critério MAP do que o critério ML.

3.4 *Soft Decoding*

Soft Decoding aplicado a códigos LDPC envolve uma relação entre o desmodulador e o descodificador mais próxima do que no caso de *hard decoding*. Em *soft decoding*, o desmodulador não devolve ao descodificador valores rígidos 0 ou 1, mas uma gama de valores intermédios que permita ao descodificador tomar uma melhor decisão.

Numa constelação 4-QAM com símbolos equiprováveis, seja a constelação e o valor recebido representada na figura 3.6.

Um sistema de recepção a funcionar com *hard decoding* retorna valores binários dependendo da localização do valor medido. Porém, desde que o valor medido esteja sempre dentro do mesmo intervalo, o desmodulador irá retornar sempre os mesmos bits. Este tipo de desmodulação impede que mais informação se conheça sobre qual foi verdadeiramente o símbolo enviado pelo transmissor e apenas se limita a verificar em qual intervalo a medição se encontra. A figura 3.6 mostra como duas medições, $S1$ e $S2$, seriam desmodulados para '10' e '11' respectivamente. No entanto, é possível constatar que o símbolo $S1$ possui um nível de certeza bastante inferior ao símbolo $S2$, assumindo que os eixos correspondem aos limites de decisão. No sentido de minimizar a

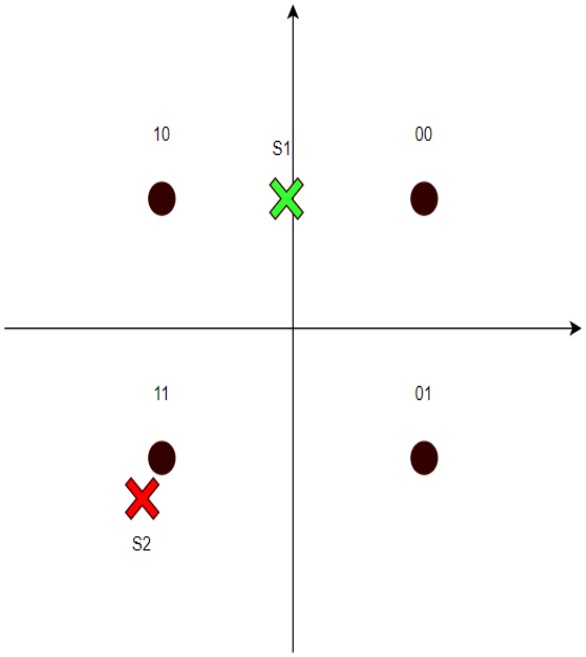

Figura 3.6: Constelação 4-QAM com duas medições de dois símbolos recebidos ambos assinalados por uma cruz.

probabilidade de haver uma desmodulação errada, pode-se retirar mais informação sobre o sistema. Quando emparelhado com códigos corretores de erros, como códigos LDPC, uma 'métrica' chamada *log-likelihood ratio* (LLR) [5] pode ser usada no momento da descodificação do código LDPC, para se obter um melhor estimativa da palavra de código transmitida do que se obteria com *hard decoding*.

3.5 LLR

A LLR, ou Log-Likelihood-Ratio, é o valor que será usado para o descodificador LDPC tomar uma melhor decisão sobre quais os símbolos que foram enviados pelo transmissor. Sendo o valor recebido r afectado pelo ruído Gaussiano, c_i o bit i da palavra de código: $(c_1, c_2, c_3, ..., c_n)$, $S_{i,j}$ o bit i do símbolo j e A_j representa a amplitude do símbolo j, então a LLR, [19], pode ser escrita como:

$$\text{llr}(c_i) = \log\left(\frac{P(c_i = 0|r)}{P(c_i = 1|r)}\right) \tag{3.21}$$

O valor da LLR indica a probabilidade do bit c_i ser 1 ou 0. No momento da decisão, caso o valor da LLR seja maior que zero, esse bit será descodificado para um 0; o bit será descodificado para um 1 caso contrário. Seja $S_{j,i} = 0$ o conjunto de símbolos onde o bit na posição i do símbolo j seja igual a zero e $S_{j,i} = 1$ o conjunto de símbolos onde o bit na posição i do símbolo j seja igual

a um. Expandindo a expressão 3.21:

$$\text{llr}(c_i) = \log\left(\frac{\sum_{s\in S_{j,i}=0} P(A_j = s|r)}{\sum_{s\in S_{i,j}=1} P(A_j = s|r)}\right) \tag{3.22}$$

Pelo teorema de Bayes:

$$\text{llr}(c_i) = \log\left(\frac{\sum_{s\in S_{i,j}=0} P(r|A_j = s)\cdot P(S_j = s)}{\sum_{s\in S_{i,j}=1} P(r|A_j = s)\cdot P(S_j = s)}\right) \tag{3.23}$$

Como exemplo, seja o valor recebido $r = 0.5 + 0.5j$, como representado na figura 3.7.

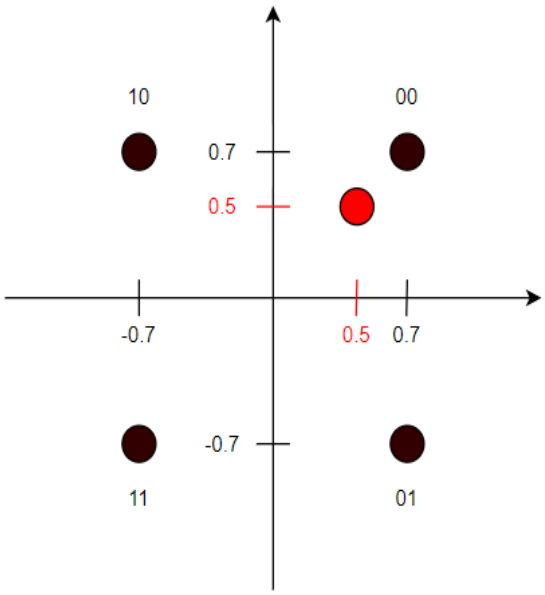

Figura 3.7: Constelação 4-QAM com sinal recebido a vermelho r= 0.5 + 0.5j

.

O cálculo da LLR para o primeiro bit, assumindo probabilidades iguais para todos os símbolos, será igual a:

$$\text{llr}(c(1)) = \log\left(\frac{P(r|A_j = (00)) + P(r|A_j = (01))}{P(r|A_j = (10)) + P(r|A_j = (11))}\right) \tag{3.24}$$

Cada termo pode ser calculado independentemente, como mostrado na expressão 3.25.

$$\begin{aligned}
P(r|A_j = (00)) &= e^{-\frac{0.2^2}{2\cdot\frac{\sigma^2}{2}}} \cdot e^{-\frac{0.2^2}{2\cdot\frac{\sigma^2}{2}}} \\
P(r|A_j = (01)) &= e^{-\frac{0.2^2}{2\cdot\frac{\sigma^2}{2}}} \cdot e^{-\frac{0.7^2}{2\cdot\frac{\sigma^2}{2}}} \\
P(r|A_j = (10)) &= e^{-\frac{0.7^2}{2\cdot\frac{\sigma^2}{2}}} \cdot e^{-\frac{0.2^2}{2\cdot\frac{\sigma^2}{2}}} \\
P(r|A_j = (11)) &= e^{-\frac{0.7^2}{2\cdot\frac{\sigma^2}{2}}} \cdot e^{-\frac{0.7^2}{2\cdot\frac{\sigma^2}{2}}}
\end{aligned} \tag{3.25}$$

De notar três aspectos:

1. O factor de normalização de uma forma de onda gaussiana $\frac{1}{\sqrt{\sigma^2 \cdot 2 \cdot \pi}}$ desaparece pois este termo é comum a todos os termos e aparece tanto no denominador como no numerador.

2. Em duas dimensões, a $P(r|p_j = s) = P_x(r|p_j = s) \cdot P_y(r|p_j = s)$, ou seja, o produto da componente x e y dará a probabilidade pretendida.

3. Visto que em QAM trabalha-se em duas dimensões, a variância segundo x e y será igual a: $\sigma^2 = \sigma_x^2 + \sigma_y^2$, no qual: $\sigma_x^2 = \sigma_y^2 = \frac{1}{2}\sigma^2$.

Veja-se que os valores das LLRs na expressão 3.23 dependem da probabilidade dos símbolos. Este aspecto será importante para o estudo de *Probabilistic Shaping* (PS) nos próximos capítulos.

3.6 *Belief propagation algorithm*

Belief propagation algorithm é o algoritmo de descodificação usado neste trabalho. Este algoritmo evolui fazendo iteração nas operações no conjunto de expressões de 3.27 até 3.29. Este algoritmo está presente em [11] e em [19].

O primeiro passo passa por calcular as LLRs dos bits constituintes da palavra de código:

$$\text{llr}(c_i) = \log\left(\frac{P(c_i = 0|r)}{P(c_i = 1|r)}\right) \tag{3.26}$$

Realizam-se depois os cálculos da cada linha de matriz de paridade $H_{paridade}$ para cada elemento. O valor i representa a coluna da matriz $H_{paridade}$, j representa a linha da matriz $H_{paridade}$.

$$L(r_{i,j}) = 2\text{atanh}\left(\prod_{i' \in V_j/i} \tanh\left(\frac{1}{2}L(q_{i',j})\right)\right) \tag{3.27}$$

$L(q_{i,j})$ para a primeira iteração será inicializado com os mesmos valores que $\text{llr}(c_i)$. Seguidamente actualizam-se os valores de $L(q_{i,j})$

$$L(q_{i,j}) = L(c_i) + \sum_{j' \in B_i/j} L(r_{ij'}) \tag{3.28}$$

Finalmente calculam-se os valores das LLRs actualizadas para os bits da palavra de código.

$$L(Q_i) = L(c_i) + \sum_{j' \in B_i} L(r_{ij'}) \tag{3.29}$$

Caso o valor de $L(Q_i)$ seja maior que zero, o bit i será descodificado para o valor binário 0, e 1, caso o contrário. Voltam-se a repetir as equações 3.27, 3.28 e 3.29 até o cálculo da síndrome retornar um vector nulo ou até um número máximo de iterações chegar ao limite. A notação V_j/i representa todos os elementos da linha j da matriz $H_{paridade}$ com excepção do elemento i e $B_{i/j}$ todos os elementos da coluna i, excepto o elemento na linha j. A figura 3.8 serve de exemplo de uma possível matriz $H_{paridade}$.

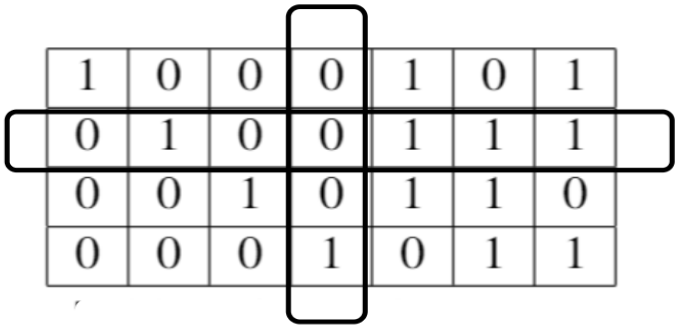

Figura 3.8: V_2 e B_4 representados por um rectângulo horizontal e vertical respectivamente. V_2 seria o vector (0100111) e B_4 seria (0001). $V_2/4$ seria o vector (010111) e $B_4/2$ seria o vector (001).

3.7 Conclusão

De acordo com a complexidade desejada, poderá optar-se pelos algoritmos apresentados de *soft decoding* e *hard decoding*. Neste trabalho, escolheu-se *soft decoding* pelo maior desempenho oferecido. Códigos longos LDPC da tecnologia DVB-S2 foram escolhidos pois permitem um método de codificação simples e ao mesmo tempo com um desempenho elevado, superando o desempenho de códigos turbo [6], que seriam a outra escolha a considerar.

Caso se decida usar *hard decoding*, o critério MAP permite calcular limites de decisão que variam de acordo com as condições do canal, permitindo um melhor desempenho que o critério ML.

Capítulo 4

Probabilistic constellation shaping

4.1 Conceito

Num canal ruidoso, a taxa de transmissão de dados tem que ser capaz de se adaptar ao valor do SNR, pois, como visto anteriormente, para haver transmissão de dados com uma taxa de erros arbitrariamente pequena, a seguinte condição terá que ser respeitada:

$$C = B \cdot \log_2(1 + SNR) \tag{4.1}$$

Assumindo uma fonte binária aleatória, a probabilidade de envio de um certo símbolo pertencente ao alfabeto da constelação será igual a qualquer outro símbolo, tal como em ASK-uniforme ou QAM-uniforme. Tecnologias como DVB-S2 [1], permitem fazer uma adaptação da taxa de transmissão de dados variando o número de pontos da constelação usada e a *code rate* usada pelos códigos corretores de erros.

A restrição do uso de símbolos com a mesma probabilidade, impõe que a entropia de uma constelação QAM uniforme esteja dentro de um intervalo discreto, como mostrado na figura 4.1.

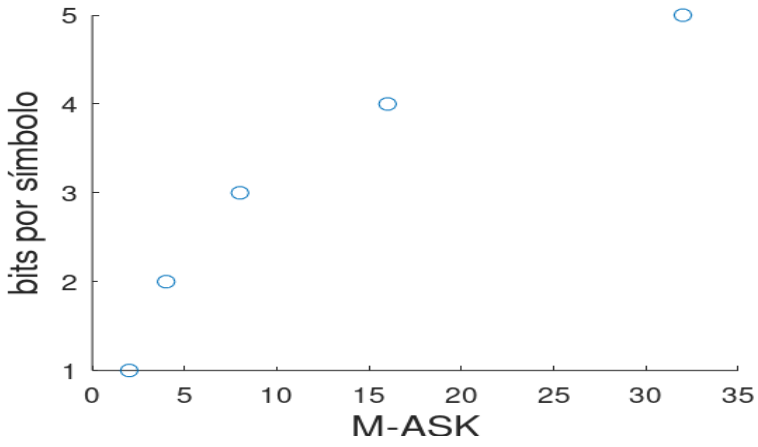

Figura 4.1: Entropia constelação M-QAM

No sentido de compensar esta grande granularidade, DVB-S2 usa códigos corretores LDPC com um vasto leque de *code rates*. Porém com *Probabilistic Shaping*, como iremos ver, é possível obter-se uma muito maior granularidade do que com adaptação da taxa de bits de informação com códigos corretores de erros.

4.2 Informação mútua

Como descrito no capítulo 2, informação mútua é descrita por:

$$I(X,Y) = H(X) - H(X|Y) \tag{4.2}$$

Sendo $H(X)$ a entropia da constelação a usar e X e Y, os símbolos enviados e recebidos respectivamente.

$H(X|Y)$ representa o valor de entropia condicionada de X sabendo Y. Esta entropia representa a quantidade de incerteza sobre X, que se obtém ao observar Y. Se X e Y forem independentes, $H(Y|X) = H(Y)$. Como neste trabalho apenas se usará ruído Gaussiano branco, o valor da entropia condicionada apenas dependerá da constelação a ser usada, e do valor do SNR. A entropia condicionada é definida por:

$$H(X|Y) = -\sum P(X,Y)\log_2 \frac{P(X,Y)}{P(Y)} \tag{4.3}$$

Num canal expresso a partir das probabilidades condicionadas, $P(X,Y)$ será expresso a partir da expressão 4.4 e $P(Y=y)$ será definida 4.5.

$$P(X=x,Y=y) = P(X=x)\cdot P(Y=y|X=x) \tag{4.4}$$

$$P(Y=y) = \sum_{x\in X} P(X=x)\cdot P(Y=y|X=x) \tag{4.5}$$

As probabilidades condicionadas $P(Y|X)$ são construídas integrando uma forma de onda gaussiana dentro dos intervalos de decisão dos respectivos símbolos. Para uma constelação 4-ASK, como representada na figura 4.2, Y, o símbolo recebido depois da introdução de ruído e X, o

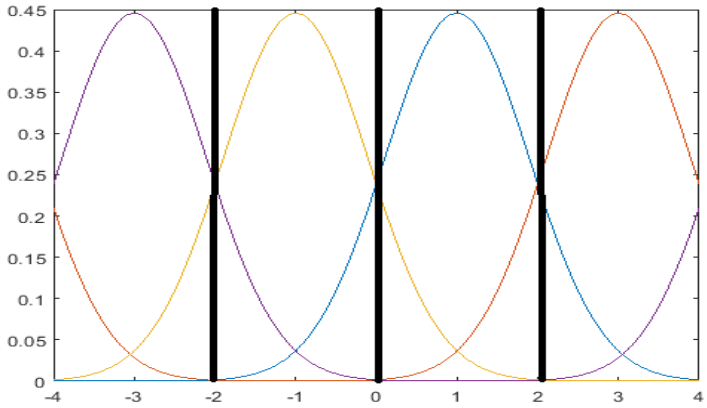

Figura 4.2: Constelação 4-ASK bipolar. As probabilidades de um símbolo ser confundido por outro símbolo no receptor corresponde à área das curvas representadas nesta imagem; os valores de decisão encontram-se representados por linhas verticais pretas.

símbolo realmente enviado no transmissor, o valor das probabilidades condicionadas para as probabilidades $P(Y=-1|X=x)$ estão representadas em 4.6.

$$\begin{aligned} P(Y=-1|X=-3) &= \int_{-2}^{0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(-3))^2}{2\sigma^2}}, dx \\ P(Y=-1|X=-1) &= \int_{-2}^{0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(-1))^2}{2\sigma^2}}, dx \\ P(Y=-1|X=+1) &= \int_{-2}^{0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(+1))^2}{2\sigma^2}}, dx \\ P(Y=-1|X=+3) &= \int_{-2}^{0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(+3))^2}{2\sigma^2}}, dx \end{aligned} \tag{4.6}$$

Tendo-se calculado as probabilidades conjuntas e probabilidades de Y, é possível calcular o valor da entropia condicionada.

De notar que o valor da entropia condicionada será tanto maior quanto menor for a distância entre os pontos da constelação, ou, quanto maior for a potência do ruído. Então para constelações com menos pontos, mantendo a mesma potência da constelação, o valor da entropia condicionada será menor. É por isso que para razões sinal-ruído mais baixas, constelações com menos pontos (para uma potência constante) deverão ser utilizadas, apesar de que, para constelações de maiores dimensões, o valor da entropia da constelação será mais elevado. A figura 4.3 representa visualmente a relação entre a entropia de uma constelação e a entropia condicionada com a informação mútua.

A partir da fórmula 4.2 pode-se notar as vantagens de *Probabilistic Constellation Shaping*: enquanto que para ASK-uniforme, o valor da entropia da constelação segue valores discretos com pouca granularidade, com um valor da entropia condicionada de acordo com o número de pontos da constelação e do SNR, PCS possui valores contínuos para o valor da entropia de uma constelação. Como irá ser visto mais à frente, com *Probabilistic Constellation Shaping*, o valor da informação mútua poderá ser maximizado especificamente para cada valor de SNR devido ao

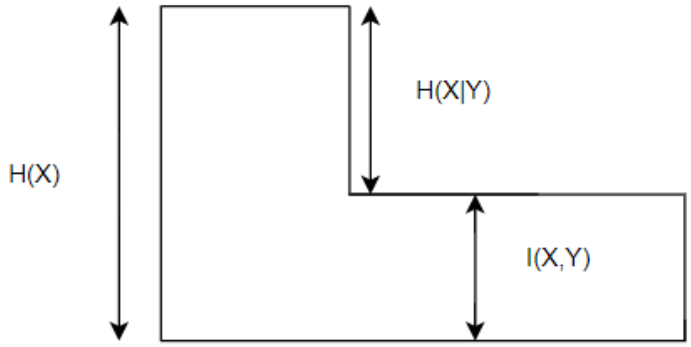

Figura 4.3: Representação gráfica de informação mútua com entropia e entropia condicionada

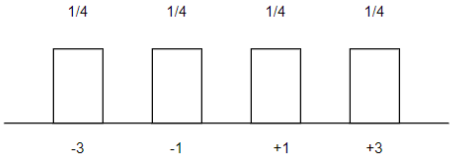

Figura 4.4: 4-ASK uniforme com entropia igual a 2 bits de informação

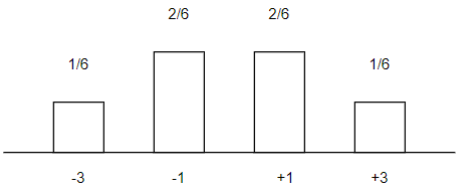

Figura 4.5: 4-ASK não uniforme com entropia aproximadamente igual a 1.65 bits de informação

facto da maior granularidade da entropia da constelação.

Nota: aquando do cálculo da informação mútua, poderão surgir valores a tender para zero. Os seguintes limites poderão ser úteis:

$$\begin{aligned} \lim_{x\to 0} x\cdot\log(\tfrac{1}{x}) &= 0 \\ \lim_{x\to 0} x\cdot\log(x) &= 0 \end{aligned} \tag{4.7}$$

4.3 Equação Maxwell-Boltzmann

Como referido na secção anterior, *Probabilistic Shaping* consegue adaptar o valor da entropia da sua constelação numa gama contínua de valores. Consegue-se isso através da libertação da restrição que, para uma fonte aleatória, os símbolos enviados são equiprováveis. *Probabilistic Constellation Shaping* tem opção de escolha de qualquer tipo de distribuição de probabilidades de símbolos que se deseje e com isso, tenciona-se obter um maior desempenho para uma gama de valores de SNR.

As figuras 4.4 e 4.5 mostra como para os mesmos pontos da constelação, é possível valores de entropia diferentes.

A distribuição escolhida irá influenciar directamente o valor da entropia da constelação, como o valor das probabilidades condicionadas, como mostrado na expressão 4.2. Porém, existe um número infinito de distribuições possíveis que se possa aplicar. Então, algum critério tem que ser usado para se escolher uma família de distribuições. Esse critério será o critério de maximização de entropia de um ponto de vista da energia da constelação. De acordo com [9], a função escolhida terá que ser capaz de respeitar às restrições a seguir:

1. Probabilidade de certo símbolo não pode ser negativo
2. A soma de todas as probabilidades tem que ser igual a 1
3. A potência total da constelação tem que igualar a uma constante definida α

Exprimindo matematicamente:

$$\begin{aligned} f(x) &\geq 0 \\ \int f(x)dx &= 1 \\ \int f(x)r_i(x)dx &= \alpha \end{aligned} \tag{4.8}$$

De acordo com [9], a função que maximiza a entropia cumprindo as restrições 4.8 é a expressão 4.9, sendo x_i pertencente ao conjunto dos pontos de constelação.

$$f(x) = \frac{e^{-v \cdot x_i^2}}{\sum_{x \in X} e^{-v \cdot x_i^2}} \tag{4.9}$$

Como se pode ver nas imagens 4.7 a 4.10, uma constelação 4-ASK como na imagem 4.6 apresentará distribuições diferentes dependendo do valor de v, segundo a expressão 4.9.

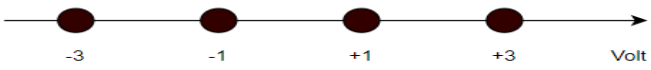

Figura 4.6: Constelação 4-ASK bipolar

Como é evidente nas figuras 4.7 a 4.10, a distribuição das probabilidades dos diversos símbolos vai ficando cada vez mais centralizado à medida que o parâmetro v vai aumentando. De facto o parâmetro v é conhecido como o *shaping parameter*, pois é este que irá definir a probabilidade de distribuição dos diversos símbolos.

Como se pode ver na figura 4.7, quando v é igual a zero, a distribuição dos diversos símbolos é a mesma de ASK uniforme. Quando se vai aumentando o valor de v, os símbolos exteriores vão sendo enviados com cada vez menos frequência. É com base neste parâmetro que se pode concluir um dos aspectos que define *Probabilistic Shaping*: para valores de v diferentes de zero, a energia da constelação baixa, em relação ao mesmo sistema ASK uniforme, com os pontos

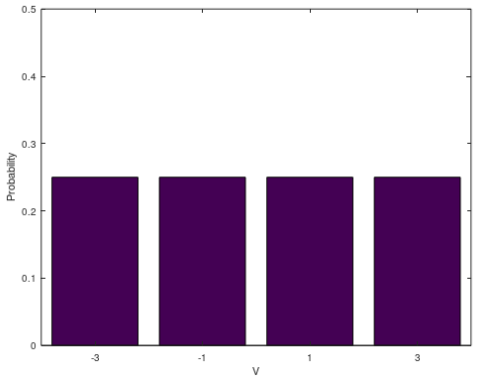

Figura 4.7: Distribuição de probabilidade de símbolos com 'v'= 0

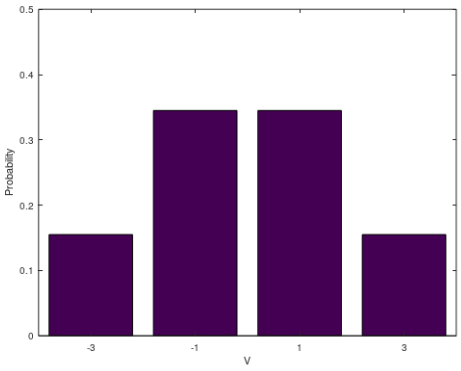

Figura 4.8: Distribuição de probabilidade de símbolos com 'v'= 0.1

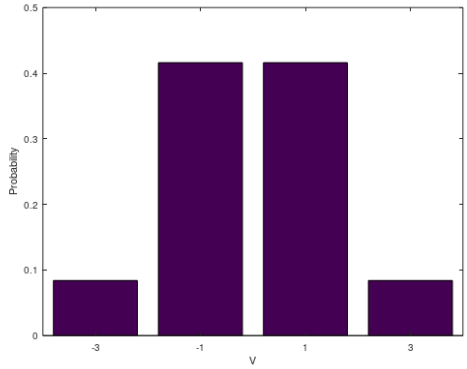

Figura 4.9: Distribuição de probabilidade de símbolos com 'v'= 0.2

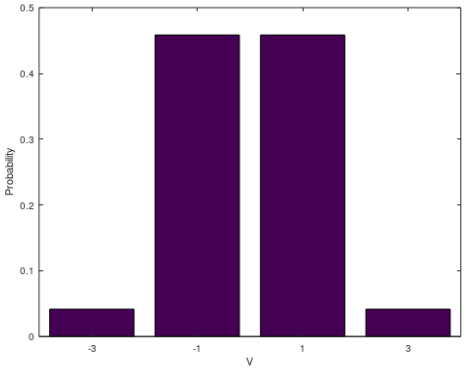

Figura 4.10: Distribuição de probabilidade de símbolos com 'v'= 0.3

da constelação na mesma posição que em PS-ASK. Isto acontece pois como os símbolos mais interiores irão ser transmitidos com maior frequência que os símbolos exteriores, a potência média da constelação cai. Para que a potência média das constelações ASK-uniforme e PS-ASK sejam constantes, terá que se fazer uma expansão da constelação PS-ASK, o que resulta numa distância entre símbolos maior no mapa de constelações. No final, para colocar a potência da constelação num valor definido, deverá ser calculado a potência actual da constelação e expandir-se todos os pontos por um factor Δ.

Por exemplo, para uma constelação 4-ASK, com alfabeto de símbolos $(-3,-1,+1,+3)$, com uma potência de constelação desejada igual a 1W, com um parâmetro de *shaping*, v, igual a 0.1, a distribuição de cada símbolo será a mesma da figura 4.8. Assim a distribuição de probabilidades de cada símbolo será: $(0.16, 0.34, 0.34, 0.16)$, respectivamente, como na figura 4.8. A potência desta constelação com a distribuição de $v = 0.1$ será igual a:

$$\begin{aligned} P_{\text{constelação}} = &\quad 0.16 \cdot |-3| + 0.34 \cdot |-1| + 0.34 \cdot |+1| + 0.16 \cdot |+3| \\ P_{\text{constelação}} = &\quad 1.64W \end{aligned} \tag{4.10}$$

Sendo pretendida uma potência média de 1 W, todos os pontos terão que ser multiplicados por um valor constante igual ao inverso do valor obtido em 4.10, igual a $1/1.64$. Desta maneira a potência da constelação é normalizada. Generalizando para todos os casos, o factor multiplicativo Δ será representado por 4.11, sendo P_{media} a potência média da constelação pretendida, $P(i)$ a probabilidade de ocorrência do símbolo i e $A(i)$, a amplitude do símbolo i.

$$\Delta = \frac{P_{media}}{\sum_{i \in I} P(i) \cdot |A(i)|} \tag{4.11}$$

4.4 Maximização da informação mútua

Probabilistic Shaping consegue alterar valores da informação mútua, como foi visto na subsecção 4.2, ao variar o parâmetro v alterando assim a entropia e distribuição dos símbolos usada. A secção 4.3 apresentou a expressão que permite calcular as distribuições para determinada constelação, a distribuição de Maxwell-Boltzmann, porém esta expressão apenas nos diz a forma da distribuição, não os valores individuais da probabilidade de cada símbolo. Este capítulo dedica-se a determinar o valor ideal do parâmetro v para maximizar a informação mútua para cada valor de SNR.

Neste trabalho comparou-se o valor da informação mútua de ASK não uniforme (*Probabilistic Shaping*) com ASK uniforme. Para maximizar os valores da informação mútua usando *Probabilistic Shaping*, é necessário conhecer os valores das probabilidades condicionadas, ou seja, as probabilidades de $P(X|Y)$, tal como o canal simétrico binário, apresentado no sub-capítulo 2.2.4. Para encontrar estes valores das probabilidades condicionadas, terá que se usar o mesmo processo descrito em 4.2. O processo de maximização é descrito a seguir.

Para uma constelação M-ASK, como por exemplo, 8-ASK, 16-ASK, etc, testa-se cada valor de v, de um conjunto de valores discretos, na equação de Maxwell-Boltzmann:

$$f(x) = \frac{e^{-v \cdot x^2}}{\sum_{x \in X} e^{-v \cdot x^2}} \tag{4.12}$$

Para cada distribuição de probabilidades de símbolos em função de v, calcula-se a potência da constelação, da mesma maneira que a expressão 4.10. Segue-se o cálculo de Δ, para fazer-se a expansão da constelação, no sentido de manter a potência da constelação igual a um valor pretendido e multiplica-se todos os pontos da constelação por Δ. Calculam-se seguidamente as probabilidades condicionadas segundo 4.6 e faz-se o cálculo da informação mútua. Dependendo do valor do SNR, a capacidade do canal será maximizada para um certo valor de v. O processo de maximização da capacidade de canal é a seguir demonstrada:

1. Escolher um valor de SNR inicial.
2. Testar um valor v pertencente a uma gama de valores.
3. Calcular as probabilidades condicionadas $P(Y|X)$ de acordo com a expressão em 4.6.
4. Calcular as probabilidades conjuntas $P(X,Y)$ com base na expressão 4.4.
5. Calcular a informação mútua com base na expressão 4.13 ou na expressão 4.2.

$$I(X,Y) = \sum_X \sum_Y P(X,Y) \log_2 \left(\frac{P(Y,X)}{P(Y) \cdot P(X)} \right) \tag{4.13}$$

Após realizar os passos de 1 a 5, consegue-se chegar ao gráfico da figura 4.11. Este gráfico foi obtido usando decisões rígidas num sistema ASK com 8 pontos (3 bits) com limites a meio da distância entre pontos de constelação adjacentes.

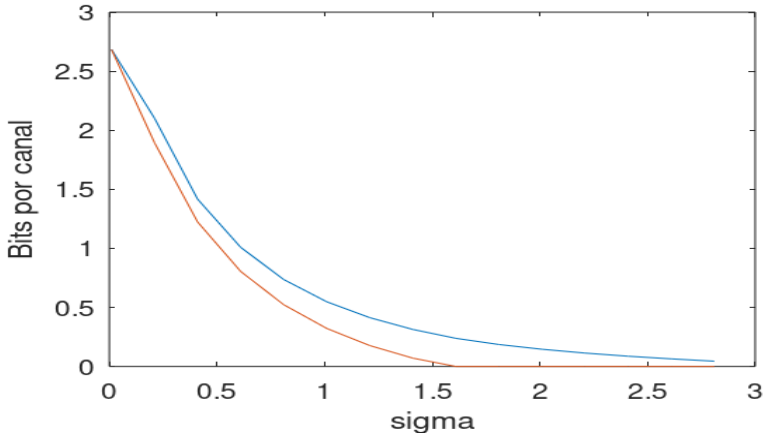

Figura 4.11: Comparação da capacidade do canal de ASK uniforme a laranja e ASK não uniforme a azul em função do desvio padrão do ruído branco σ.

4.5 Conclusão

Probabilistic Shaping combinado com ASK consegue obter melhores valores da capacidade de canal para cada valor do desvio padrão do ruído comparado com ASK uniforme. Estes dados mostram que existe vantagens ao usar *Probabilistic Shaping*. ASK uniforme parece conseguir uma granularidade tão grande como PS-ASK, porém na prática, com um uso de um número limitados de constelações e de códigos correctores de erros, ASK-uniforme irá conseguir adaptar a taxa de dados de informação dependendo do valor do SNR, mas sem a granularidade de PS-ASK.

Capítulo 5

Implementação de *Probabilistic Shaping*

A implementação de *Probabilistic Constellation Shaping* depende de um algoritmo que consiga transformar um conjunto de bits aleatório, num conjunto de símbolos com uma probabilidade definida. E para ser possível aplicar códigos correctores de erros, sem alterar a amplitude dos símbolos, será necessário uma arquitectura única que difere de ASK uniforme. Será então analisado um algoritmo de *Distribution Matcher* e a arquitectura do sistema.

5.1 *Distribution Matcher*

Uma fonte aleatória binária sem memória produz zeros e uns com igual probabilidade. Estes bits têm que ser mapeados num conjunto de símbolos com uma distribuição pretendida. O algoritmo que faz essa tradução de bits para símbolos é o *Distribution Matcher* (DM). Vários tipos de *Distribution Matchers* foram publicados na literatura, tais como F2V (*fixed to variable*) *Distribution Matchers* [3], ou *Sphere Shaping* [12]. F2V *Distribution Matchers* sofrem de propagação de erros entre blocos [25]. Em [24], são apresentados diversos DMs. Um DM deve ser capaz de mapear bits aleatórios para uma sequência de símbolos que seja unicamente descodificável no receptor.

5.1.1 *Distribution Matcher* baseado em dicionários

Um *Distribution Matcher* pode ser um algoritmo que mapeia uma sequência de bits para uma sequência de símbolos através de um dicionário em memória. Desta maneira pode obter-se distribuições de probabilidades pretendidas. Para a distribuição de probabilidade de símbolo representada na tabela 5.1, pode existir uma tabela de conversão que converta bits em símbolos com a probabilidade escolhida, como indicado na tabela 5.2.

Para ser possível codificar todas as sequências de bits em símbolos, exigindo que a sequência seja unicamente descodificável, é necessário garantir que o número de sequências de símbolos seja

Símbolo	Probabilidade
A	3/4
B	1/4

Tabela 5.1: Probabilidade de símbolo

maior ou igual ao número de sequências de bits. Sendo k_c o tamanho do bloco de bits a codificar, exprimindo matematicamente:

$$k_c \leq \log_2(N_{\text{sequências}}) \tag{5.1}$$

Como se pode ver na tabela 5.2, a codificação de dois bits necessita do envio de uma sequência de quatro símbolos o que acaba por resultar numa taxa de bits de $2/4 = 0.5$ bits / símbolo. O objectivo do DM é codificar bits em símbolos de uma maneira que seja unicamente descodificável no receptor.

Porém, note-se que a entropia dos símbolos da tabela 5.1 será aproximadamente igual a 0.811 bits por símbolo. Na tabela 5.2, cada dois bits corresponde a uma sequência de quatro símbolos, o que resulta numa eficiência de $2/4 = 0.5$ bits por símbolo, ou seja, este DM sofre de uma penalização de $0.811 - 0.5 = 0.311$ bits de informação por símbolo. Para um número de bits k_c no qual se faz corresponder uma sequência de símbolos com tamanho n_c, a taxa de dados em bits/símbolo, de um DM será igual a:

$$R_{\text{dm}} = \frac{k_c}{n_c} \text{ bits/símbolo} \tag{5.2}$$

No sentido de aumentar o desempenho de um DM, pode-se aumentar o tamanho da sequência de símbolos. Seja a tabela 5.3 representativa das probabilidades de ocorrência de cada símbolo.

O número de sequências possíveis de símbolos dada por esta distribuição será igual ao produto das combinações possíveis dos respectivos símbolos. Admitindo que se usa um bloco de símbolos com tamanho de 10 símbolos, com a mesma distribuição da tabela 5.3, existirá um símbolo 'A', dois símbolos 'B', cinco 'C' e dois 'D'. O número de combinações em que se podem combinar todos estes símbolos será igual a:

$$N_{\text{sequências}} = \binom{10}{1} \cdot \binom{9}{2} \cdot \binom{7}{5} \cdot \binom{2}{2} = 7560 \tag{5.3}$$

Bits	Sequência
00	AAAB
01	AABA
10	ABAA
11	BAAA

Tabela 5.2: Tabela de conversão bits para símbolos

Símbolo	Probabilidade
A	1/10
B	2/10
C	5/10
D	2/10

Tabela 5.3: Tabela com probabilidades de ocorrência de cada símbolo

O número de bits que podem ser codificados será dado pela expressão 5.1, ou seja, $\log_2(7560) = 12.884$ bits. Como apenas se pode ter valores inteiros de bits, o número de bits que se pode codificar com este exemplo será igual a 12 bits. A taxa de dados que se pode obter neste caso será igual a 12/10= 1.2 bits/símbolo.

Se agora o número de símbolos num bloco fosse de 20 símbolos, o número de combinações possível seria igual a 5.8×10^8. O número de bits que seria possível codificar com um bloco de 20 símbolos com a distribuição de 5.3 seria 30 bits, o que resulta num número de bits/símbolo de 1.45, superior em 0.25 bits/símbolo do que no caso anterior. Ao aumentar o tamanho do bloco de símbolos a codificar, é possível obter-se a figura do gráfico 5.1

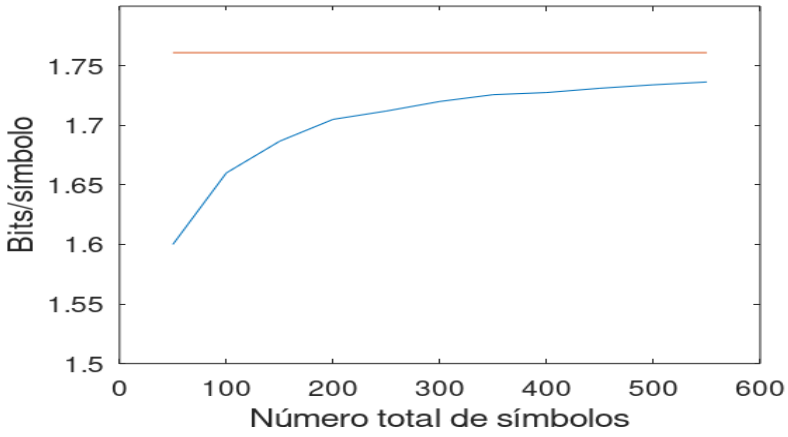

Figura 5.1: Bits/símbolo em função do tamanho do número de símbolos a azul. A laranja está representado o limite máximo teórico

O gráfico 5.1 representa um comportamento essencial num *Distribution Matcher* construído sob a forma de um dicionário, descrita acima: a penalização da taxa de bits por símbolo (obtida pela diferença da linha laranja com a linha azul na figura 5.1) será tanto menor, quanto maior for o tamanho da sequência de símbolos e por consequência, o tamanho do bloco de bits. Isto leva a concluir que para um *Distribution Matcher* atingir o seu desempenho máximo (valor dado pela linha laranja na imagem 5.1) o tamanho do bloco de símbolos tem que tender para o infinito.

Na figura 5.1, está representado um limite teórico que define a taxa máxima permitida para o *Distribution Matcher*: isso é o valor da entropia da constelação. A entropia da constelação, usando

as mesmas probabilidades que na tabela 5.3 será definida por:

$$\begin{aligned} H(A) = & -\left(\tfrac{1}{10}\log_2\left(\tfrac{1}{10}\right)+\tfrac{2}{10}\log_2\left(\tfrac{2}{10}\right)+\tfrac{5}{10}\log_2\left(\tfrac{5}{10}\right)+\tfrac{2}{10}\log_2\left(\tfrac{2}{10}\right)\right) \\ H(A) = & \quad 1.76 \text{ bits/símbolo} \end{aligned} \tag{5.4}$$

Como se pode ver no gráfico 5.1, os ganhos são cada vez mais pequenos à medida que o tamanho do número de símbolos vai aumentando. Desde que o tamanho de blocos de símbolo seja suficientemente grande, a penalização obtida pelo *Distribution Matcher* pode ser desprezada. Ao usar-se blocos com tamanho de 550 símbolos com uma distribuição igual à tabela 5.3, a taxa de bits por símbolo será igual a 1.737 bits / símbolo, que comparando com o valor da entropia, 1.76, isso significa que para um bloco de 550 símbolos, existe uma penalização de 0.023 bits/símbolo.

De acordo com [25], sendo n o tamanho do bloco de símbolos, k o número de símbolos diferentes que constituem o bloco de símbolos e $H(A)$ a entropia da distribuição dos símbolos, o número de possíveis sequências de símbolos pode ser escrita por:

$$N_{\text{sequências}} >= \binom{n+k-1}{k-1}^{-1} \cdot 2^{n\cdot H(A)} >= (n+k)^{-k} 2^{n\cdot H(A)} \tag{5.5}$$

Tomando o logaritmo de base 2:

$$\frac{\log_2(N_{\text{sequências}})}{n} \geq \frac{-k\cdot\log_2(n+k)}{n} + H(A) \tag{5.6}$$

A taxa de dados será igual a:

$$R_{dm} \geq \frac{-k\cdot\log_2(n+k)}{n} + H(A) - 1/n \tag{5.7}$$

Todas estas equações foram retiradas de [25]. Estas expressões mostram que quando n tende para infinito, a taxa de dados do *Distribution Matcher* tenderá para a entropia da constelação, ou seja:

$$\lim_{n\to\infty} R_{dm} = H(A) \tag{5.8}$$

Também se pode mostrar que a divergência de Kullback–Leibler definida na sub-secção 2.3, artigo [6], tende para zero à medida que o bloco de símbolos tende para infinito. Este ponto é importante pois, com um tamanho de blocos de símbolos finito, existe um conjunto discreto de distribuições de probabilidades possíveis. Por exemplo, com um bloco de símbolos finito, não é possível fazer com que um símbolo tenha uma probabilidade de ocorrência com um valor não racional. Como estas distribuições podem ser diferentes das distribuições dadas pela expressão 4.12, deverão ser usados blocos grandes de símbolos para minimizar a divergência entre as duas distribuições, ou seja, para tornar estas duas distribuições o mais idênticas possíveis.

Para fazer *distribution matching* com base num dicionário, basta apenas escrever todas as sequências de bits com tamanho k_c, com todas as sequências de símbolos com tamanho n_c e guardar em memória. À medida que os bits vão aparecendo, vai-se codificando com base no dicionário e obtém-se a sequência de símbolos pretendida. Porém à medida que o bloco de símbolos vai aumentando, a quantidade de memória necessária para guardar todas as sequências de símbolos aumenta rapidamente, o que torna esta abordagem impraticável. Apenas como exemplo, para um tamanho de blocos de 100 símbolos, o número de sequências distintas com a distribuição da equação 5.3 será aproximadamente $1.43 \cdot 10^{50}$. Neste trabalho irão usar-se blocos maiores que 5000 símbolos, o que mostra que não é prático guardar em memória todas as sequências possíveis.

5.1.2 *Constant composition distribution matching*

Constant composition distribution matching (CCDM), foi introduzido em [25]. Este algoritmo permite através de cálculos codificar uma sequência de bits em sequências de símbolos e vice versa. Este algoritmo pertence à família F2F, o que significa que a propagação de erros só ocorrerá no bloco no qual os símbolos estão representados. Este algoritmo vem resolver o problema do DM baseado em dicionários apresentado na secção anterior de maneira que irá ser discutido neste capítulo.

CCDM partilha muitas similaridades com o algoritmo de compressão apresentado da subsecção 2.2.2, sendo por esse motivo que os algoritmos que irão ser discutidos para CCDM, neste trabalho, provêm de técnicas de implementação de *aritmetic encoding*.

CCDM funciona ao variar as probabilidades de ocorrência de cada símbolo à medida que novos símbolos vão sendo escolhidos. À medida que os símbolos vão sendo escolhidos, a probabilidade desse símbolo irá descer até chegar a zero. Uma analogia usada nesta sub-secção vai ser o de comparar as probabilidades de ocorrência de símbolos à probabilidade de se retirar uma bola com um certo índice de um conjunto de n_{total} de bolas. A probabilidade de ocorrência de uma bola com índice i será igual a:

$$P_i = \frac{n_i}{n_{\text{total}}} \tag{5.9}$$

sendo n_i o número de bolas com o índice i e n_{total}, o número total de bolas.

Usando a analogia das bolas, a probabilidade de ocorrência de uma bola i, será igual ao número de bolas que partilham o mesmo índice i a dividir pelo número de bolas total, independentemente do índice. Tal como em *arithmetic encoding*, irão ser definidos intervalos numa escala de zero a um, proporcionais à probabilidade de ocorrência de uma bola com índice i. Estes intervalos não se sobrepõem e a soma de todos os intervalos será igual a um. No momento da escolha de uma bola, irão ser criados novos intervalos dentro dos intervalos da bola escolhida. Porém, à medida que certa bola com índice i vai sendo escolhida, o número de bolas com esse índice é decrementado por 1. O mesmo acontece com o número de bolas total. A cada iteração do algoritmo, as probabilidades de ocorrência de certa bola terão que ser novamente calculadas, de acordo com a equação 5.9. Quando todas as bolas de todos os índices forem escolhidas, escolher-se-á um número dentro

do intervalo obtido.

Como exemplo, sendo o número de bolas de cada índice representado na tabela 5.4, o algoritmo decorrerá como na figura 5.2.

Indice da bola	Número de bolas índice i	Probabilidade
A	1	1/8
B	2	2/8
C	5	5/8

Tabela 5.4: Número de bolas de índice 'i' para um tamanho do bloco de 8 bolas.

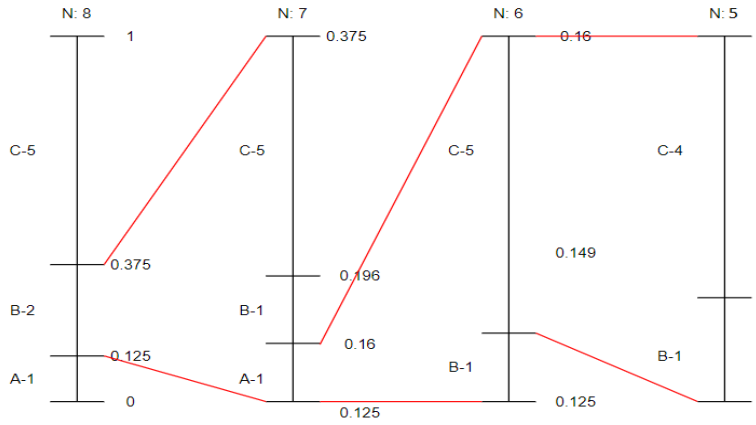

Figura 5.2: Algoritmo CCDM. O índice da bola e o número de bolas desse índice estão representadas à esquerda dos intervalos. Por exemplo "C-5"significa que existem 5 bolas com índice 'C'. Acima do gráfico, é indicado o número total de bolas, ou seja, N:8 indica que existem um número total de 8 bolas. De notar que as probabilidades de ocorrência de cada bola alteram-se a cada iteração. Inicialmente, a bola com índice 'A' terá uma probabilidade de ocorrência de 1/8, porém na segunda iteração, a probabilidade de ocorrência desta bola será igual a 1/7, pois, como a bola com índice 'B' foi seleccionada, o número total de bolas baixou de oito para sete.

Na figura 5.2, a sequência a codificar foi: (B,A,C). Como se pode observar na figura, quando numa primeira interacção o símbolo B foi escolhido, o número de bolas desse índice baixou, reduzindo o seu intervalo na seguinte iteração. Quando todas as bolas tiverem sido esgotadas, pode-se atribuir um número identificador na sequência, como em *arithmetic encoding*. O receptor faz a descodificação realizando as operações inversas, o que envolve saber anteriormente o número de bolas de cada índice, e como os intervalos das bolas estão organizados.

A implementação do algoritmo de CCDM é baseada no artigo [26]. O pseudocódigo, baseado na linguagem de programação C, está representado a seguir.

```
1  void bits2symbols(uint64_t message)
2  {
3      uint64_t high= MAXVALUE;
4      uint64_t low= 0;
5      uint64_t range;
6      uint64_t aux;
7      uint64_t cumulative[2];
8
9      struct Probabilities{
10         int freq[124];
11         int n_symbols;
12     };
13
14     char c;
15     struct Probabilities probability;
16
17     initialize_symbols(&probability);
18     for (i=1;i<n_symbols;i++)
19         {
20             range= high - low;
21             aux= (message - low)/range;
22             c= getSymbol(aux, &cumulative, &probability);
23             update_symbol_probabilities(&probability);
24             high= low + range*cumulative[0];
25             low=  low + range*cumulative[1];
26         }
27 }
```

O código apresentado é responsável por transformar uma sequência de bits em símbolos. As variáveis *high* e *low* representam os limites superiores e inferiores, respectivamente, dos símbolos, como apresentado na figura 5.2. Inicialmente a variável *high* é inicializada ao valor máximo, que neste caso corresponde a $2^{64} - 1$ devido ao uso de uma variável de 64 bit (*uint64_t*) e a variável *low* é inicializada a zero; *aux* apenas é usada como uma variável intermédia para facilitar a leitura do código. A variável *message* contém a sequência de bits que se quer transformar em símbolos e *range* representa a distância de *high* a *low*. Tendo como referência a figura 5.2, e assumindo que o valor, em binário, da sequência de bits que se quer codificar é o valor 0.15625, inicialmente *high* e *low* seriam inicializados a *MAXVALUE* e zero, respectivamente, sendo que depois de uma primeira iteração, os valores *high* e *low* passariam a representar os valores 0.375 e 0.125, respectivamente. A função *getSymbol* é responsável por duas tarefas fundamentais: a primeira passa por retornar os símbolos representativos da sequência de bits que se quer codificar, o que neste exemplo, numa primeira iteração do algoritmo, retornaria o símbolo "B"como visto na figura 5.2; e a segunda passa por retornar as probabilidades acumulativas dos símbolos que precedem o símbolo descodificado (neste caso, como o símbolo descodificado foi o símbolo "B", o único símbolo que o antecede, como visto na figura 5.2 pela ordem crescente na escala de 0 a 1, é o símbolo "A"). Esta variável é representada por *cumulative* e é composta por dois valores diferentes: *cumulative[0]* serve para

apenas guardar a acumulativa das probabilidades dos símbolos anteriores e *cumulative[1]*, serve para guarda a acumulativa das probabilidades dos símbolos anteriores e do próprio símbolo, ou seja, esta variável, segundo a figura 5.2, numa primeira iteração, *cumulative[0]* corresponde à probabilidade do símbolo "A" e *cumulative[1]* corresponde à probabilidade do símbolo "A" mais a probabilidade do símbolo que realmente foi descodificado (o símbolo "B"). Se hipoteticamente, o valor a descodificar fosse, numa primeira iteração, um valor correspondente ao símbolo "C", o valor de *cumulative[0]* seria a soma das probabilidades dos símbolos "A" e "B" e *cumulative[1]* seria a soma das probabilidades de "A", "B" e "C". No final é feito uma actualização das probabilidades dos símbolos o que significa que, voltando à analogia das bolas presentes num conjunto de n_{total} de bolas, ao número de bolas com o índice "B", retira-se uma bola e decrementa-se também em uma unidade o número de bolas total.

Apresenta-se agora o código responsável por transformar uma sequência de símbolos numa sequência de bits. Todas as variáveis servem o mesmo propósito que no código apresentado anteriormente, excepto que, *getSymbol* lê o símbolo que foi recebido e guarda-o em *c* e *findIntervalValues* devolve as probabilidades acumulativas representativas do símbolo em questão guardadas na variável *c* (as probabilidades acumulativas são guardadas em *cumulative[0]* e *cumulative[1]*, como descritos anteriormente). As inicializações das variáveis foram omitidas neste código pois são as mesmas apresentados o código anterior.

```
1
2     for (i=1;i<nSymbols;i++)
3     {
4         c= getSymbol();
5         findIntervalValues(c, &cumulative, &probability);
6         update_symbol_probabilities(&probability);
7         range= high - low;
8         high= low + range*cumulative[0];
9         low=  low + range*cumulative[1];
10    }
```

No final de todas as iterações, escolhe-se um valor dentro dos limites *high* e *low*. Neste trabalho, o valor escolhido corresponde ao valor da variável *high*. De notar que o valor escolhido tem que ter, em binário, um número de bits específico, especificado na sub-secção 5.1. Este valor, assumindo que foram recebidos todos os símbolos sem erros, será o valor que foi de facto enviado. Como este valor será guardado em binário, facilmente se obtêm os bits que constituem este valor, obtendo-se assim a sequência de bits que o receptor transmitiu.

O código apresentado, embora funcional, apresenta um problema grave: o uso de uma variável com precisão finita. Como dito anteriormente, a declaração de uma variável "`uint64_t`" não possui precisão suficiente para mapear uma sequência maior que 64 bits (imagine-se que se pretende executar o algoritmo numa sequência de bits de tamanho 100). Duas soluções foram analisadas para solucionar este problema. A primeira solução passa por usar variáveis com uma

Valor decimal	Valor em binário
0.375	00111110110000000000000000000000
0.125	00111110000000000000000000000000

Tabela 5.5: Representação binário e decimal de 0.375 e 0.125

precisão variável de bits. Usou-se então uma biblioteca chamada *GNU Multiple Precision Arithmetic Library*, a qual pode ser encontrada em [2]. Esta solução está longe de uma solução ideal mas foi a única solução funcional. O problema advém de que, ao usar uma sequência de bits de, por exemplo, 1000 bits é necessário que as variáveis usadas nos códigos acima apresentados tenham uma precisão maior que 1000 bits e, como se pode ver no código, existem operações de soma, subtracção, multiplicação e divisão que são demasiado computacionalmente intensivas numa variável com um tal precisão. A segunda solução (que não produziu um código funcional, ver explicação mais à frente) passa por usar variáveis com uma precisão "pequena", de apenas 64 bits e usar alguns truques para evitar o problema de precisão insuficiente. Para isso, estudou-se a implementação apresentada em [26], no capítulo "*A PROGRAM FOR ARITHMETIC CODING*".

A ideia desta solução é que, à medida que sucessivas iterações do algoritmo vão sendo executadas, certos bits podem ser descartados pois não acrescentam informação nova. Olhe-se mais uma vez para a figura 5.2. Na primeira iteração do algoritmo, tanto a variável *high* e *low*, representados pelos valores 0.375 e 0.125 têm algo em comum: o primeiro bit será um 0. Veja-se a tabela 5.5.

Sendo que 0.375 e 0.125 têm o primeiro bit igual, para além da primeira iteração do algoritmo, este bit não nos dá mais informação, sendo que pode ser descartado. Ao ser descartado, usando uma variável com uma precisão finita de bits (como por exemplo 64 bits), consegue-se acrescentar um espaço para um novo bit da sequência de bits que se quer codificar. O método funcionaria por, ao retirar o primeiro bit, todos os restantes bits seriam deslocados uma posição para a esquerda, e acrescenta-se um novo bit na posição mais à direita. De notar que, em binário, uma multiplicação por dois causa que todos os bits sejam deslocados uma posição para a esquerda, o que na prática significa que retira-se o primeiro bit, multiplica-se o valor resultante por dois e soma-se o bit que se quer acrescentar. O código que transforma uma sequência de bits em símbolos está representada a seguir.

```
1  void bits2symbols(message)
2  {
3      fillMessage(&message);
4      for (i=1;i<n_symbols;i++)
5          {
6              range= high - low;
7              aux= (message - low)/range;
8              c= getSymbol(aux, &cumulative, &probability);
9              update_symbol_probabilities(&probability);
10             high= low + range*cumulative[0];
11             low=  low + range*cumulative[1];
12
```

```
13              for (i=1;i<n_symbols;i++)
14          {
15              range= high - low;
16              aux= (message - low)/range;
17              c= getSymbol(aux, &cumulative, &probability);
18              update_symbol_probabilities(&probability);
19
20              for (;;) {
21                  if (high < HALF) {
22                      //Do nothing
23                  }
24                  else if (low >= HALF) {
25                      message -= HALF;
26                      low -= HALF;
27                      high -= HALF;
28                  }
29
30                  else if (low>=FIRST_QTR && high < THIRD_QTR) {
31                      message -= FIRST_QTR;
32                      low -= FIRST_QTR;
33                      high -= FIRST_QTR;
34                  }
35                  else break;
36
37                  low= low * 2;
38                  high= high * 2;
39                  message= 2 * message + input_bit();
40
41              }
42          }
43      }
44  }
```

Como se pode ver, da linha 20 até à linha 41 é onde se faz o escalonamento das variáveis *high*, *low* e *message*. Este escalonamento permite que as variáveis *high* e *low* nunca tenham valores tão próximos uma da outra que, devido a uma precisão finita das variáveis em uso, sejam representadas pelos mesmos valores em binário, bloqueando assim o algoritmo. Neste algoritmo, três novas variáveis são introduzidas: *HALF*, é responsável por guardar o valor de metade da escala que se está a usar, ou seja, se estiver a usar-se variáveis de 64 bits, *HALF* teria o valor $(2^{64}-1)/2$; *FIRST_QTR* representa um quarto do valor máximo que se está a usar, ou seja, $(2^{64}-1)/4$ e *THIRD_QTR*, representa três quartos do valor máximo, ou seja, $(2^{64}-1)\times 3/4$. De notar que estas variáveis são multiplicadas por dois nas linhas 37 a 39 e que, neste algoritmo, a variável *message* é uma variável com uma precisão finita sendo que, a cada iteração do algoritmo, é introduzido um bit na posição de bit menos significativa pela função *input_bit*. O código que transforma os símbolos em bits está representada a seguir.

```
1
2     for (i=1;i<nSymbols;i++)
3     {
4         c= getSymbol();
5         findIntervalValues(c, &cumulative, &probability);
6         update_symbol_probabilities(&probability);
7         range= high - low;
8         high= low + range*cumulative[0];
9         low=  low + range*cumulative[1];
10
11        for (;;) {
12                if (high < HALF) {
13                    bits_to_follow= outputBits(0, bits_to_follow);
14                }
15                else if (low >= HALF) {
16                    bits_to_follow= outputBits(1, bits_to_follow);
17                    low -= HALF;
18                    high -= HALF;
19                }
20
21                else if (low>=FIRST_QTR && high < THIRD_QTR) {
22                    bits_to_follow++;
23                    low -= FIRST_QTR;
24                    high -= FIRST_QTR;
25                }
26                else break;
27
28                low= low * 2;
29                high= high * 2;
30            }
31    }
```

Como se pode ver, este código partilha o mesmo escalonamento de variáveis que no código anterior. Neste caso, é necessário transformar símbolos em bits de modo que, sabendo que tanto as variáveis *high* e *low* se encontram acima de *HALF*, o primeiro bit destas duas variáveis é garantidamente um 1. Caso estas variáveis se encontrem abaixo de *HALF*, o primeiro bit será um 0. Estes bits vão sendo retirados via a função *outputBits*. O problema surge quando *high* e *low* se encontram dentro do terceiro quarto e do segundo quarto, respectivamente. Neste caso, a variável *high* tem um primeiro bit como um 1 e *low* como um 0, o que gera conflito sobre qual o verdadeiro bit a descodificar. Na verdade, apenas se pode saber qual o verdadeiro bit a descodificar no futuro, após as próximas iterações. Porém, parece ser aqui o problema que impede que o código seja funcional. Como dito anteriormente, este código usando variáveis com precisão finita não funciona e pensa-se que é na condição em que *high* e *low* estão dentro do terceiro e segundo quarto, respectivemente, que o problema surge. É preciso trabalho adicional para averiguar onde estão os problemas existentes ou mesmo para verificar se a solução apresentada é ou não possível de se

executar. Talvez esta abordagem não seja correcta e outra abordagem terá que ser desenvolvida. A dificuldade de encontrar o erro deve-se ao facto de ser um algoritmo iterativo, que apenas mostra um erro no final de, por exemplo, vinte iterações. *Debugg* neste caso é difícil e os erros podem ser difíceis de justificar. Um deles por exemplo, parte do princípio que, ao fazer contas manualmente, uma variável, por exemplo a *high*, deva dar um valor igual a *HALF*, mas por alguma razão, *high* encontra-se uma unidade abaixo deste limite, o que causa uma propagação de erros nas próximas iterações massiva, eliminado a funcionalidade do algoritmo. Como este algoritmo não é funcional, partes do código foram omitidas e simplificadas por motivos de leitura. Caso se queira encontrar mais informação, dirijo a sua atenção para [26]. Questões de *overflow* e *underflow* são respondidas, bem como outros problemas que surgem com o uso de variáveis com uma precisão finita. Devido ao facto que o único algoritmo funcional (o algoritmo que depende do uso da biblioteca *GNU Multiple Precision Arithmetic Library*) não ser óptimo devido à complexidade computacional envolvida, nas simulações realizadas no capítulo 6 não se usou nenhum algoritmo de DM. Ao invés de se gerar bits aleatórios para depois serem codificados em símbolos com uma distribuição pretendida, gerou-se imediatamente símbolos com essa mesma distribuição. O capítulo 6 fornece mais informações sobre o método usado para gerar os resultados.

5.2 Arquitectura do sistema

5.2.1 *Probabilistic Amplitude Shaping*

Probabilistic Amplitude Shaping [6] é a arquitectura capaz de realizar *Probabilistic Shaping* sem corromper a distribuição dos símbolos a enviar. Nesta arquitectura, os símbolos irão ser divididos em duas partes: amplitude e sinal. Num sistema ASK a funcionar com m bits por símbolo, $m-1$ bits de cada símbolo serão dedicados à amplitude do sinal, que é dada pelos símbolos do DM, e um bit será dado pelo código LDPC, como mostrado na figura 5.3.

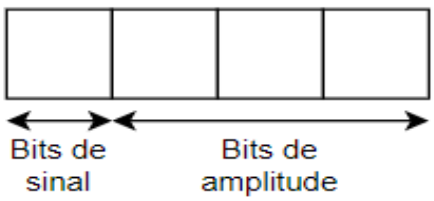

Figura 5.3: Formato de um símbolo de quatro bits da arquitectura *Probabilistic Amplitude Shaping*. O primeiro bit determina o sinal do símbolo e os restantes três bits, a amplitude do símbolo.

.

Assumindo um gerador aleatório binário, o DM transformaria esses bits em símbolos com uma probabilidade definida, como mostrado na figura 5.4.

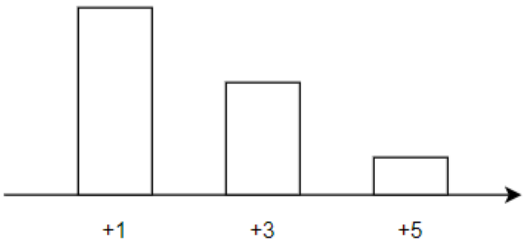

Figura 5.4: Amplitude de sinal

.

Os símbolos do DM serão transformados numa representação binária e usados para gerar os bits de paridade do código LDPC. Estes bits de paridade serão responsáveis por gerar o sinal dos símbolos (negativo ou positivo) consoante forem zeros ou uns. De notar que para uma constelação possuir uma potência média constante, todos os pontos da constelação terão que ser multiplicados por Δ, como mostrado na secção 4.3. O diagrama de blocos do sistema está representado na figura 5.5.

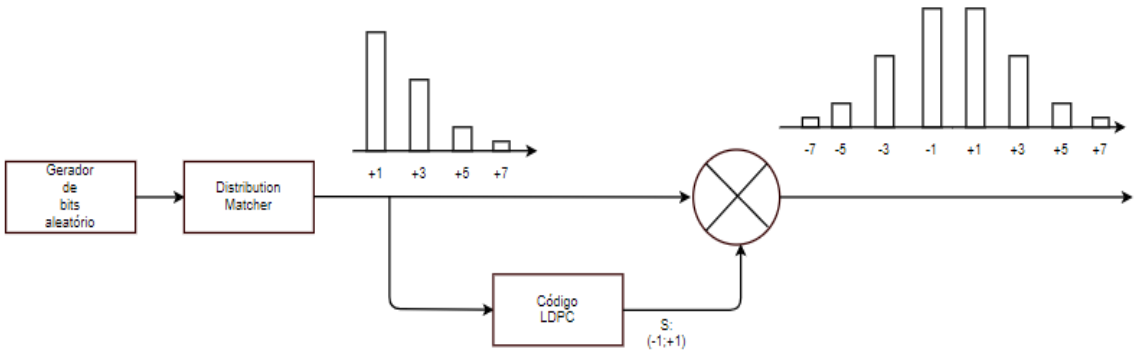

Figura 5.5: Arquitectura de *Probabilistic Shaping*

Como se pode ver na figura 5.3, a *Code Rate* desta arquitectura está limitada a $(m-1)/m$. No caso da figura referida, existem três bits de amplitude para um bit de sinal, fazendo com que a *code rate* seja de $3/4$. Porém, como neste trabalho se usaram códigos LDPC da tecnologia DVB-S2, as *code rates* estão limitadas a: (1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10), o que não torna possível usar alguns destes códigos. Para superar este problema, a solução passa por usar alguns dos bits do próprio gerador de bits aleatório, e usá-los para definir directamente o sinal das amplitude do sinal, função que anteriormente era exclusivamente feita pelos bits de paridade do código LDPC. De notar que neste esquema aplicam-se bits de paridade aos símbolos, não aos bits que serão enviados como no caso de ASK uniforme, pois é imperativo que o *Distribution Matcher* receba correctamente todos os símbolos que tenham sido enviados. Caso contrário, existirá propagação de erros por todo o bloco de símbolos. Sendo $H(A)$ a entropia de símbolos do *distribution*

matcher, a taxa de bits de informação por símbolo desta arquitectura será igual a:

$$R = H(A) + \Gamma, \tag{5.10}$$

sendo c a *code rate* do código LDPC usada e m, o número de bits por símbolo usada na modulação ASK, Γ será igual a:

$$\Gamma = 1 - (1 - c)m \tag{5.11}$$

Neste trabalho, usou-se mapeamento de Gray como descrito na secção 2.1.4. Descreve-se agora como se usou este mapeamento em conjunto com *Probabilistic Amplitude Shaping*. O tipo de mapeamento usado (mapeamento de Gray) é simétrico em relação ao ponto intermédio se se ignorar o primeiro bit de cada símbolo, como se pode ver na figura 5.6. Vendo a imagem referida da esquerda para a direita, o primeiro símbolo (ignorando o primeiro bit de todos os símbolos) é o símbolo '00' que é igual ao último símbolo: '00'. O segundo símbolo: '01' é igual ao penúltimo símbolo: '01'. O mesmo acontece para todos os símbolos. Esta abordagem permite separar os bits de amplitude dos bits de sinal, pois, neste mapeamento, o primeiro bit de cada símbolo determina o sinal do símbolo, ou seja, se o símbolo pertence à metade positiva ou negativa da constelação ASK. Isto significa que o DM, para o caso da figura 5.6, produz os símbolos: (00,01,10,11) de acordo com uma distribuição escolhida, sendo que estes símbolos são depois complementados com os bits de paridade que são os bits de sinal. O facto deste mapeamento ser simétrico é importante pois, como em PCS, os símbolos são enviados com uma distribuição simétrica, tal como mostrado na figura 5.5, o DM apenas se limita a produzir os símbolos: (00,01,10,11) e o gerador de bits de paridade apenas se limita a completar os símbolos com os bits de sinal. Desta maneira é possível aplicar códigos corretores de erros aos símbolos e manter a distribuição pretendida dos símbolos.

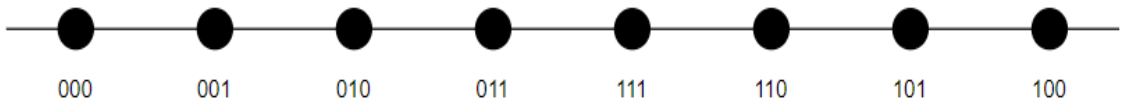

Figura 5.6: Mapeamento de Gray para 8-ASK

5.3 Conclusão

Apresentou-se um algoritmo possível para implementação de um *Distribution Matcher* pelo algoritmo de *Constant Composition Distribution Matcher*. A implementação eficiente deste algoritmo irá depender do tamanho de blocos usado, que será tanto mais eficiente quanto maior o tamanho do bloco de símbolos. Como referido, a única implementação do algoritmo funcional não é óptima, pelo que será necessário um maior estudo para se obter uma implementação mais eficiente.

A arquitectura apresentada consegue resolver dois problemas de *Probabilistic Shaping*: como manter a amplitude dos símbolos e ao mesmo tempo, aplicar códigos correctores de erros. Esta arquitectura não permite *code rates* menores que $\frac{m-1}{m}$, porém, este aspecto não representa um problema pois não é necessário que sejam os códigos correctores de erros a fazer adaptação da taxa de dados porque a própria constelação já o faz.

Capítulo 6

Resultados de simulação

Aplicou-se a arquitectura da figura 5.5 com mapeamento de Gray em ambiente MATLAB no sentido de se obter dados de simulação. O *Distribution Matcher* não foi simulado por motivos de velocidade de simulação sendo gerado directamente símbolos com uma distribuição pretendida em vez de bits aleatórios, o que significa que a penalização da presença do *Distribution Matcher* foi ignorada. Porém, como se usou blocos de símbolos grandes (acima de 1000 símbolos), a penalização poderá ser desprezada, como mostrado na secção 5.1, tornando os resultados apresentados válidos.

Numa primeira fase simulou-se o comportamento do sistema com a introdução de ruído gaussiano branco. Compararam-se os melhores resultados obtidos com ASK uniforme com ASK não uniforme. Os resultados podem ser vistos na tabela 6.1 e na figura 6.1.

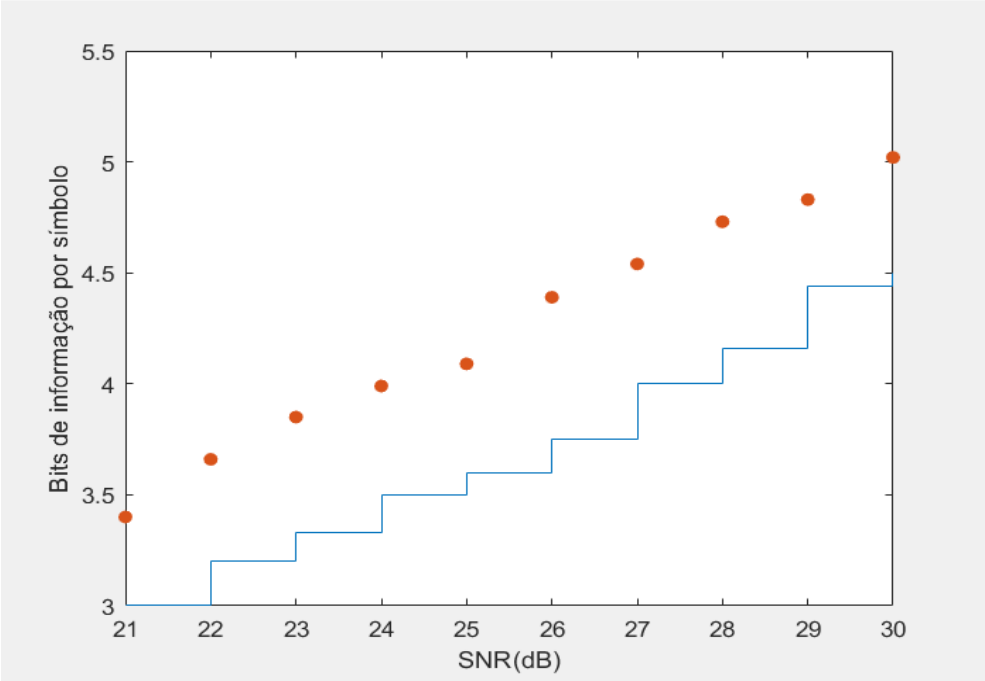

Figura 6.1: Bits de informação por canal em função de SNR. A vermelho ASK não uniforme; a azul ASK uniforme

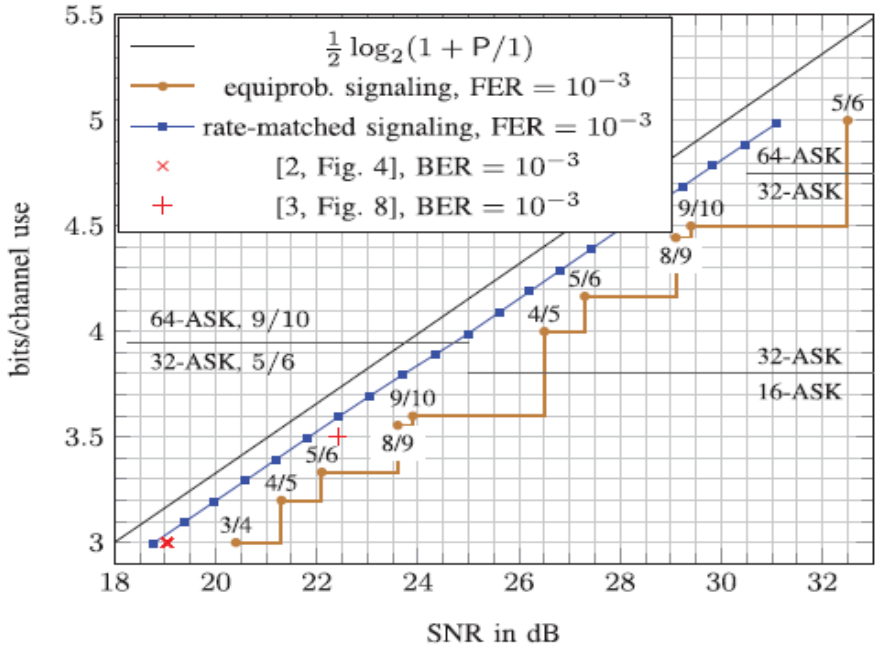

Figura 6.2: Resultados teóricos obtidos em [6]

Os resultados obtidos por simulação foram semelhantes ao resultados obtidos em [6], como representado na figura 6.2. As diferenças de resultados podem explicar-se pelo uso de um método de descodificação de códigos LDPC mais computacionalmente exigente neste trabalho e em [6], o método de avaliação foi para uma FER menor que 10^{-3} enquanto que neste trabalho usou-se como critério uma BER inferior a 10^{-4}. A figura 6.1 representa as vantagens de *Probabilistic Shaping*: um maior número de bits de informação por símbolo em todos os valores de SNR testados do que no caso de ASK não uniforme.

A taxa de dados que será possível enviar, sendo R a quantidade de bits de informação por símbolo definido em 5.10 e R_s a taxa de símbolos, será igual a:

$$R_{sistema} = R_s \cdot R \text{ bits de informação/segundo} \tag{6.1}$$

Seguidamente comparou-se o comportamento de ASK uniforme com ASK não uniforme simulando o comportamento de uma fibra de óptica da autoria de [15]. Este código já foi usado em [15] e [17] de modos que não se questiona a sua validade. Simulou-se um sistema coerente com troços de fibra de comprimento de 10 km, com um amplificador de fibra dopada com Erbium entre cada troço, como representado na figura 6.3.

O índice de refracção do núcleo foi simulado com o valor 1.4538 e a bainha com o valor 1.439185. Simulou-se a propagação de um sinal óptico no comprimento de onda de 1.55 μm e apenas se usou uma polarização. Definiu-se uma atenuação de 0.2 dB/km onde foram incluídos efeitos não lineares via *Kerr effect*, cujo valor foi colocado a $0.0013\,\mathrm{W}^{-1}\cdot\mathrm{m}^{-1}$. O método numérico implementado foi baseado no método de "*split-step Fourier*"e inclui compensação da dispersão no domínio eléctrico. São usados amplificadores ópticos EDFA (*Erbium-doped fiber*

	ASK Uniforme			ASK Não Uniforme			
SNR dB	Ordem modulação ASK	Code Rate	Bits de informação por símbolo	Ordem modulação ASK	Code Rate	Bits de informação por símbolo	Parâmetro v
21	16	3/4	3	32	5/6	3.4	0.006
22	16	4/5	3.2	32	5/6	3.66	0.0035
23	16	5/6	3.33	32	5/6	3.85	0.0028
24	16	8/9	3.5	32	5/6	3.99	0.0019
25	16	9/10	3.6	32	5/6	4.09	0.011
26	32	3/4	3.75	64	9/10	4.39	0.0021
27	32	4/5	4	64	9/10	4.54	0.0017
28	32	5/6	4.16	64	9/10	4.73	0.0013
29	32	8/9	4.44	64	9/10	4.83	0.0011
30	32	9/10	4.5	64	9/10	5.02	0.0008

Tabela 6.1: Dados de simulação dos melhores resultados de ASK uniforme e ASK não uniforme com ruído Gaussiano branco para uma BER inferior a 10^{-4}. As *code rate*s testadas foram: (3/4, 4/5, 5/6, 8/9, 9/10).

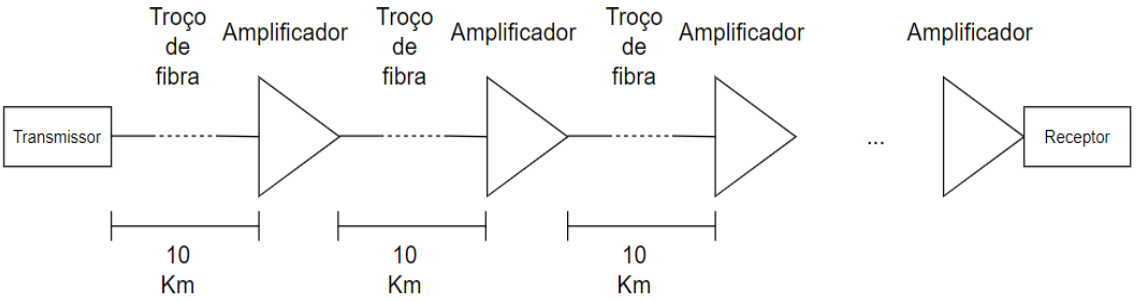

Figura 6.3: Esquemático do sistema simulado.

amplifier) dimensionados para regenerar a potência inicial injectada. Estes módulos são simulados com ruído ASE (*Amplified spontaneous emission*) e com isso cada amplificador possui uma figura de ruído de 5 dB. Todos os valores foram analisados para uma BER menor que 10^{-4}. Primeiramente foram comparados os valores obtidos em ASK-uniforme com PS-ASK em alguns cenários de simulação. Os resultados estão tabelados nas tabelas 6.2 e 6.3 e representados graficamente nas figuras 6.4 e 6.5, respectivamente.

Os valores de bits de informação por símbolo variam consoante a potência média injectada na fibra. Usaram-se diferentes valores de potências injectadas na fibra no sentido de identificar a potência ideal a ser injectada. Os resultados estão representados na imagem 6.6. Para potências mais elevadas, fenómenos de distorção não lineares começam a prevalecer, prejudicando o desempenho do sistema. A imagem 6.7 representa o funcionamento do sistema para uma potência mais elevada de 10 dBm. Pode-se observar um deterioramento da quantidade de bits de informação por sím-

	ASK Uniforme			ASK PS			
Distância (km)	Code Rate	Entropia	Ordem ASK	Code Rate	Entropia	Parâmetro v	Ordem ASK
100	5/6	3.33	16	9/10	3.73	0.0060	32
200	3/4	3.00	16	9/10	3.30	0.011	32
300	8/9	2.67	8	9/10	3.03	0.13	16
400	5/6	2.50	8	9/10	2.83	0.024	16
500	4/5	2.40	8	9/10	2.67	0.030	16
600	3/4	2.25	8	9/10	2.54	0.036	16
700	9/10	1.80	4	9/10	2.45	0.041	16
800	9/10	1.80	4	9/10	2.32	0.051	8
900	8/9	1.78	4	9/10	2.25	0.059	8
1000	8/9	1.78	4	9/10	2.17	0.067	8

Tabela 6.2: Comparação ASK-uniforme com ASK-PS. Potência injectada média de 6 dBm com 20×10^9 símbolos/segundo.

	ASK Uniforme			ASK PS			
Distância (km)	Code Rate	Entropia	Ordem ASK	Code Rate	Entropia	Parâmetro v	Ordem ASK
100	9/10	3.60	16	9/10	4.38	0.0015	32
200	9/10	3.60	16	9/10	3.99	0.0040	32
300	5/6	3.33	16	9/10	3.72	0.0061	32
400	4/5	3.20	16	9/10	3.48	0.0085	32
500	3/4	3.00	16	9/10	3.30	0.011	32
600	9/10	2.70	8	9/10	3.14	0.015	16
700	9/10	2.70	8	9/10	2.99	0.019	16
800	5/6	2.50	8	9/10	2.72	0.028	16
900	5/6	2.50	8	9/10	2.67	0.012	8
1000	5/6	2.50	8	9/10	2.64	0.017	8

Tabela 6.3: Comparação ASK-uniforme com ASK-PS. Potência injectada média de 8 dBm com 20×10^9 símbolos/segundo.

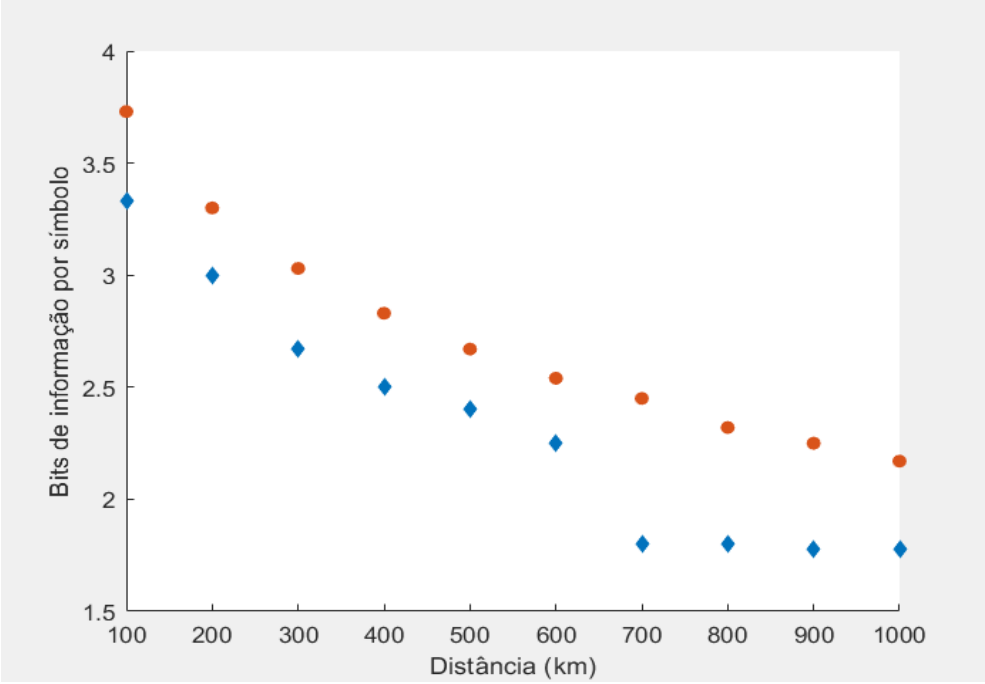

Figura 6.4: Comparação de ASK-PS (bolas vermelhas) e ASK-uniforme (diamantes azuis). Potência injectada média é de 6 dBm com uma taxa de símbolos de 20×10^9 símbolos por segundo.

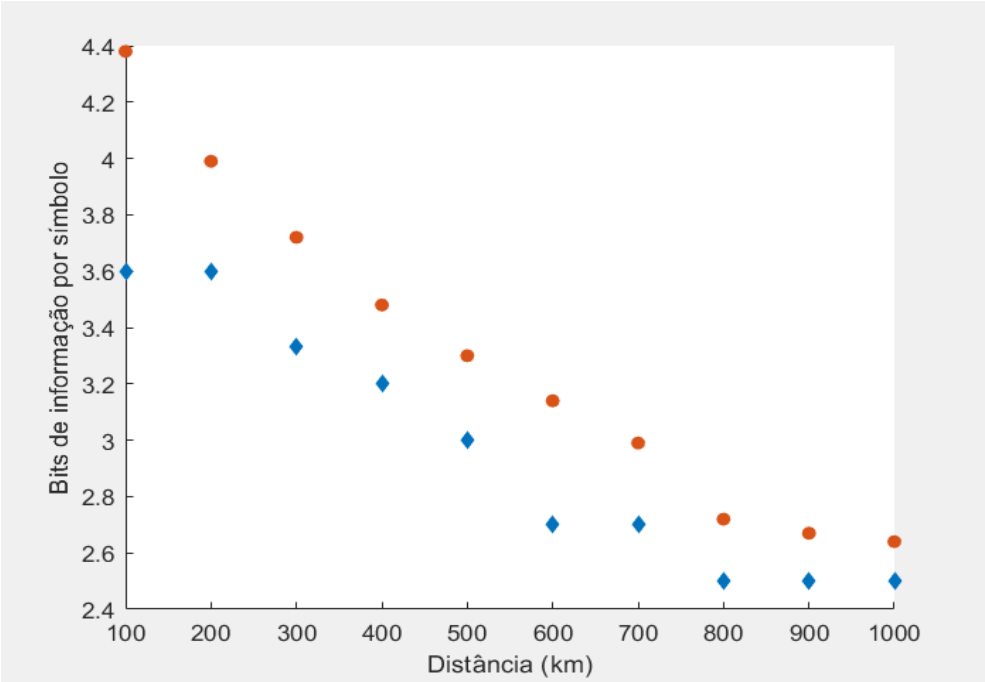

Figura 6.5: Comparação de ASK-PS (bolas vermelhas) e ASK-uniforme (diamantes azuis). Potência injectada média é de 8 dBm com uma taxa de símbolos de 20×10^9 símbolos por segundo.

bolo, principalmente na região entre os 300 km e 800 km, no qual o uso de uma potência inferior acaba por resultar num melhor desempenho do que com o uso de uma potência injectada superior.

Para potências ainda mais elevadas, como 12 dBm, os fenómenos de distorção não lineares são ainda mais relevantes, levando à conclusão que, para as condições testadas, a potência injectada óptima situa-se entre os 8 dBm e 10 dBm. Com uma potência injectada de 12 dBm, os melhores valores obtidos dos 100 km aos 500 km, são por esta ordem: $(4.00, 3.60, 2.70, 2.70, 1.80)$ bits de informação por símbolo, fazendo com que estas taxas de dados sejam mais baixas do que no caso de uma potência injectada de 8 dBm ou 10 dBm, principalmente à medida que a distância aumenta.

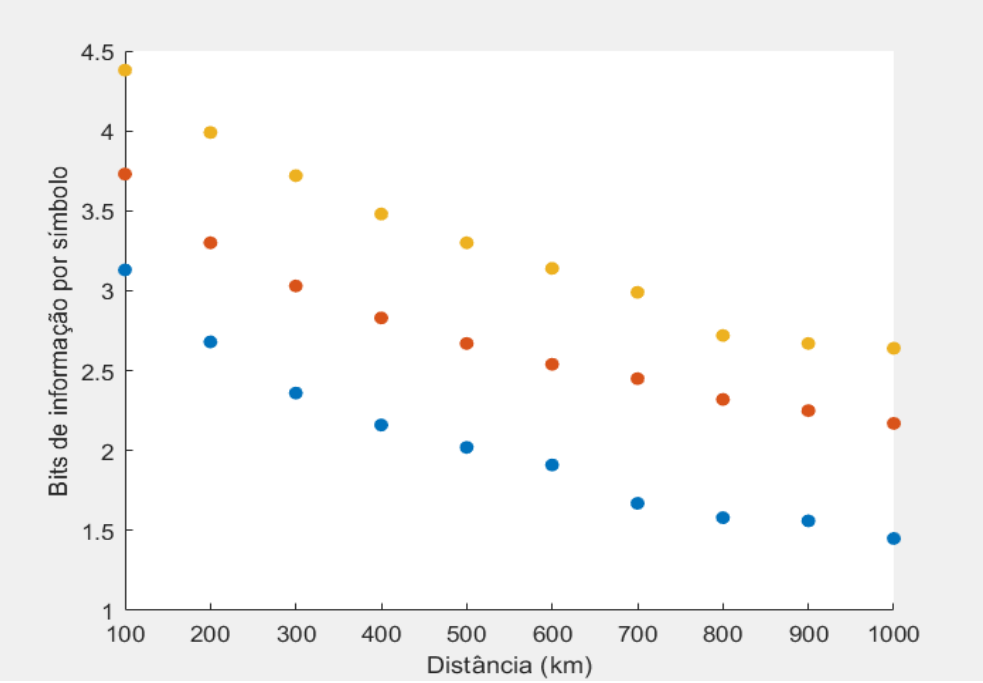

Figura 6.6: Comparação de várias potências médias injectadas usando ASK_PS. Os pontos inferiores a azul representam uma potência média injectada de 4 dBm; os pontos intermédios a vermelho representam uma potência de 6 dBm e a amarelo, os pontos superiores, uma potência média injectada de 8 dBm.

É possível ainda aumentar a taxa de bits de informação por símbolo baixando a taxa de símbolos por segundo no sistema. Ao reduzir a taxa de símbolos, prevê-se um aumento de bits de informação por símbolo devido ao ser enviado um pulso mais largo, fazendo com que os efeitos de distorção acabem por ter menos impacto relativamente a um impulso mais curto. Estas previsões são justificadas na figura 6.8. De notar que, embora o sistema com uma taxa de símbolos de símbolos de 20×10^9 possua uma taxa de bits de informação por símbolo menor que o sistema com uma taxa de símbolos de 10×10^9, o facto de que o último funciona a metade da taxa de símbolos por segundo faz com que o primeiro, num sentido de velocidade de transmissão de dados, seja um sistema mais desejado, pois embora cada símbolo contenha menos informação, como consegue o dobro dos símbolos, acaba por ser um sistema com maior capacidade de transmissão de informação.

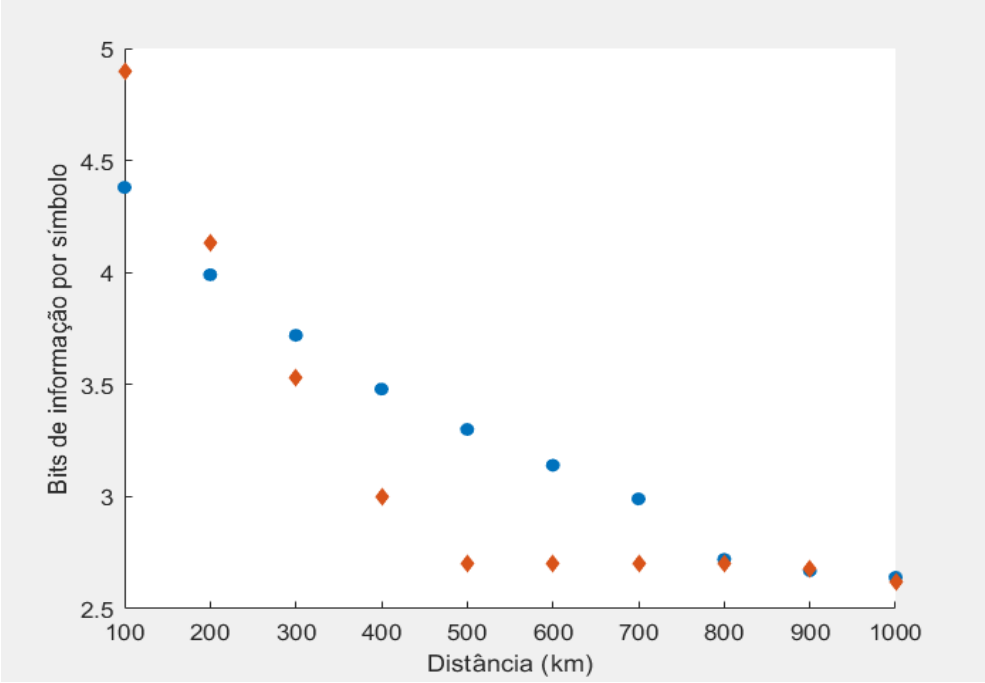

Figura 6.7: Comparação do funcionamento do sistema para uma potência de 8 dBm (bolas azuis) e 10 dBm (diamantes vermelhos)

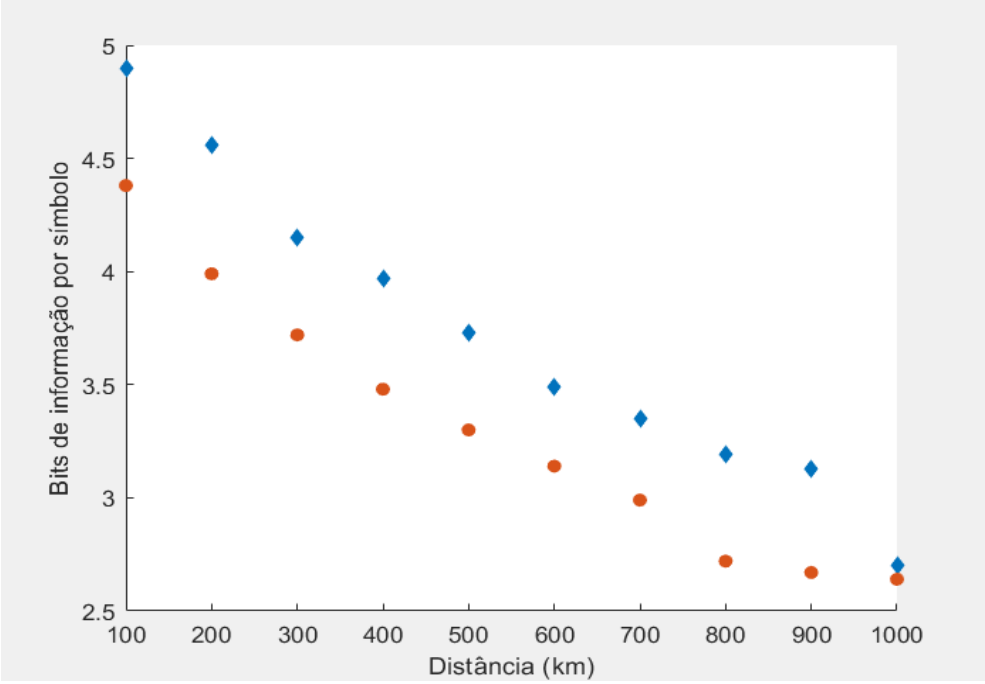

Figura 6.8: Desempenho do sistema para uma potência de 8 dBm comparando uma taxa de símbolos de 10×10^9 símbolos por segundo (diamantes azuis) com um sistema com uma taxa de símbolos de 20×10^9 símbolos por segundo (bolas vermelhas)

6.1 Conclusão

Usando a arquitectura de *Probabilitic Amplitude Shaping*, nas condições testadas, foi possível obter-se uma melhoria da taxa de bits de informação a rondar entre cinco e dez por cento, em relação ao número total de pontos da constelação em ASK-uniforme. A adaptação da taxa de bits de informação em PS acompanha as condições do canal, mostrando a flexibilidade de PS às condições de canal. Como mostrado, a taxa de bits de informação por símbolos vai aumentando à medida que a potência injectada vai subindo até um limite, no qual efeitos não lineares começam a deteriorar o desempenho do sistema que faz com que tais potências não se tornem desejadas. O uso de uma taxa de símbolos menor também contribui para um aumento de bits de informação por símbolo, porém embora uma menor taxa de símbolos torne o sistema mais robusto a efeitos de ruído conseguindo-se usar uma maior eficiência de bits de informação por símbolo, essa maior eficiência pode não ser capaz de compensar pela menor taxa de símbolos, como mostrado na última simulação acima. No geral, mostrou-se que ASK-PS consegue aumentar a taxa de dados tanto em sistemas dominados por ruído branco, como em fibras ópticas.

Capítulo 7

Conclusões e trabalho futuro

Neste trabalho foi desenvolvido um sistema ASK com *Probabilistic Constellation Shaping* e posteriormente fez-se a sua comparação com um sistema ASK uniforme. Foi possível verificar que em todas as condições analisadas, ASK-PS superou a taxa de bits de informação para uma BER fixa comparando com ASK-uniforme. O uso de técnicas como as LLRs e o algoritmo *Belief Propagation* como apresentadas neste trabalho são bastante pesadas computacionalmente. Simplificações destes algoritmos, são possíveis com uma redução de desempenho, como realizado em [6].

Uma melhor implementação do DM também poderá ser desenvolvida. Tal como mostrado, o único algoritmo de DM funcional não é prático de implementar. Um melhor algoritmo tornaria ASK-PS uma técnica prática de implementar.

Para um sistema ASK-PS onde as condições de canal variassem ao longo do tempo, teriam também que ser desenvolvidas técnicas de adaptação de canal, um pouco como em DVB-S2. Um possível problema que pode surgir é que, em PCS, o número de combinações de *code rates*, número de pontos de constelação diferentes e valores do parâmetro v é bastante elevado dependendo de quão grande ou pequena se pretende que a granularidade seja. Uma ideia seria testar previamente quais as melhores combinações para determinado valor de SNR e tabelar esses valores, fazendo com que não seja preciso calcular a melhor combinação possível em tempo real. Porém, provavelmente seria possível desenvolver uma técnica mais eficaz do que a descrita.

A implementação de um sistema QAM-PS é directamente aplicável através das técnicas apresentadas neste trabalho. Ao usar duas constelações ASK-PS ortogonais entre si, QAM-PS seria possível.

Em geral, um aumento de 5% a 10% da taxa de bits de informação de ASK-PS em relação a ASK-uniforme será possível de se obter, caso a complexidade acrescida de *hardware* necessária não seja um factor decisivo.

Referências

[1] Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVB-S2 Extensions (DVB-S2X). https://www.etsi.org/deliver/etsi_en/302300_302399/30230702/01.03.01_20/en_30230702v010301a.pdf. [Acedido pela última vez em 22/06/2021].

[2] The GNU Multiple Precision Arithmetic Library. https://gmplib.org.

[3] Rana Ali Amjad e Georg Böcherer. Fixed-to-variable length distribution matching. Em *IEEE International Symposium on Information Theory*.

[4] H. Baher e J. Coffey. On the design of digital nyquist channel filters. *IEEE Transactions on Circuits and Systems*, 33(4):460–462, 1986.

[5] Claude Berrou. Codes and turbo codes. https://link.springer.com/content/pdf/10.1007%2F978-2-8178-0039-4_9.pdf. [Acedido pela última vez em 22/06/2021].

[6] Georg Böcherer, Fabian Steiner, e Patrick Schulte. Bandwidth efficient and rate-matched low-density parity-check coded modulation. *IEEE Transactions on Communications*, 63(12):4651–4665, 2015.

[7] J. Cai, M. V. Mazurczyk, H. G. Batshon, M. Paskov, C. R. Davidson, Y. Hu, O. V. Sinkin, M. A. Bolshtyansky, D. G. Foursa, e A. N. Pilipetskii. Performance comparison of probabilistically shaped qam formats and hybrid shaped apsk formats with coded modulation. *Journal of Lightwave Technology*, 38(12):3280–3288, 2020.

[8] Junho Cho e Peter J. Winzer. Probabilistic constellation shaping for optical fiber communications. *Journal of Lightwave Technology*, 37(6):1590–1607, 2019.

[9] T. M. Cover e 2nd Ed J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, chapter 12, 1988.

[10] Ivan Djordjevic. Chapter 6 - classical error correcting codes. Em Ivan Djordjevic, organizador, *Quantum Information Processing and Quantum Error Correction*, páginas 175–225. Academic Press, Oxford, 2012.

[11] Robert G. Gallager. Low-Density Parity-Check Codes. https://web.stanford.edu/class/ee388/papers/ldpc.pdf. [Acedido pela última vez em 22/06/2021].

[12] Yunus Can Gültekin, Tobias Fehenberger, Alex Alvarado, e Frans M. J. Willems. Probabilistic shaping for finite blocklengths: Distribution matching and sphere shaping. *Entropy*, 22(5):581, May 2020.

[13] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[14] P.G. Howard e J.S. Vitter. Analysis of arithmetic coding for data compression. Em *[1991] Proceedings. Data Compression Conference*, páginas 3–12, 1991.

[15] L. M. Pessoa J. S. Tavares e H. M. Salgado. Nonlinear compensation assessment in few-mode fibers via phase-conjugated twin waves. *Journal of Lightwave Technology (JLT)*, 2017.

[16] Rinu Jose e Ameenudeen Pe. Analysis of hard decision and soft decision decoding algorithms of ldpc codes in awgn. Em *2015 IEEE International Advance Computing Conference (IACC)*, páginas 430–435, 2015.

[17] H. M. Salgado L. M. Pessoa e I. Darwazeh. Performance evaluation of phase estimation algorithms in equalized coherent optical systems. *in IEEE Photonics Technology Letters*, 21(17):1181–1183, 2009.

[18] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, e D.A. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2):585–598, 2001.

[19] MATLAB. Low-density parity-check (LDPC) decoding. https://www.mathworks.com/help/5g/ref/nrldpcdecode.html. [Acedido pela última vez em 22/06/2021].

[20] Ali Mirani, Erik Agrell, e Magnus Karlsson. Low-complexity geometric shaping. *Journal of Lightwave Technology*, 39(2):363–371, 2021.

[21] John C. Porcello. Designing and implementing low density parity check (ldpc) decoders using fpgas. Em *2014 IEEE Aerospace Conference*, páginas 1–7, 2014.

[22] Gautham Prasad, Haniph A. Latchman, Youngjoon Lee, e Weiler A. Finamore. A comparative performance study of ldpc and turbo codes for realistic plc channels. Em *18th IEEE International Symposium on Power Line Communications and Its Applications*, páginas 202–207, 2014.

[23] J. G. Proakis e M. Salehi. *Communications Systems Engineering*. Prentice Hall, 2nd Edition, 2002.

[24] Patrick Schulte. Algorithms for Distribution Matching. https://mediatum.ub.tum.de/doc/1534053/1534053.pdf. [Acedido pela última vez em 22/06/2021].

[25] Patrick Schulte e Georg Böcherer. Constant composition distribution matching. *IEEE Transactions on Information Theory*, 62(1):430–434, 2016.

[26] IAN H. WIllEN, RADFORD M. NEAL, and JOHN G. CLEARY. ARITHMETIC CODING FOR DATA COMPRESSION. https://web.stanford.edu/class/ee398a/handouts/papers/WittenACM87ArithmCoding.pdf. [Acedido pela última vez em 22/06/2021].

[27] Michael Zoltowski. Equations for the Raised Cosine and Square-Root Raised Cosine Shapes. http://www.commsys.isy.liu.se/TSKS04/lectures/3/MichaelZoltowski_SquareRootRaisedCosine.pdf. [Acedido pela última vez em 22/06/2021].