AVALIAÇÃO DE DESEMPENHO DE CÓDIGOS TURBO EM AMBIENTES COM RUÍDOS IMPULSIVOS

AUTOR: MARCELO ROBERTO LOPES

TRABALHO DE GRADUAÇÃO DO CURSO DE ENGENHARIA DE INFORMAÇÃO

Orientador: Prof. Dr. Murilo Bellezoni Loiola

Santo André, 24 de agosto de 2015.

MARCELO ROBERTO LOPES

AVALIAÇÃO DE DESEMPENHO DE CÓDIGOS TURBO EM AMBIENTES COM RUÍDOS IMPULSIVOS

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel em Engenharia de Informação da Universidade Federal do ABC

Prof. Orientador: Dr. Murilo Bellezoni Loiola

Santo André, 2015

AGRADECIMENTOS

Agradeço a todos que me incentivaram durante o curso de graduação, com palavras de apoio e amizade.

À minha família pelo apoio e carinho.

Aos meus professores do curso de Engenharia de Informação da UFABC, pelo apoio, incentivo e conselhos úteis.

Ao professor orientador Murilo Loiola, pela disponibilidade em apoiar e pela paciência e amizade.

Aos meus pais,

Ozório “In Memorian” e Cida

RESUMO

Este trabalho trata da análise de desempenho de códigos corretores de erros avançados, concentrando-se nos códigos turbo, em sistemas de comunicação cujo canal degrada o sinal transmitido pela presença de ruído impulsivo. O modelo escolhido para a representação estatística do ruído impulsivo é o baseado na distribuição de probabilidade alfa-estável simétrica e a análise de desempenho do código corretor de erro é feita através de simulação computacional, verificando a taxa de erro de bit em função da relação sinal-ruído no receptor.

ABSTRACT

This work deals with performance analysis of advanced error correcting codes, focusing on turbo codes, when they are applied to communication systems whose channel is impaired by impulsive noise. The model chosen for the statistical representation of the impulsive noise is based on symmetric alpha-stable distribution and performance analysis of the error correcting code is done by computer simulation by checking the bit error rate as a function of the signal-to-noise ratio at the receiver.

LISTA DE FIGURAS

ABREVIATURAS

AWGN:	*Additive White Gaussian Noise* - Ruído Aditivo Branco e Gaussiano
BCJR:	*Bahl, Cocke, Jelinek e Raviv*
BER:	*Bit Error Rate* - Taxa de erro de bit
BPSK:	*Binary Phase Shift Keying* - Chaveamento por deslocamento de fase binário
E_b/N_0:	Razão entre a energia de bit e a densidade espectral de potência do ruído
IIR:	*Infinite Impulse Response* - Resposta ao impulso finita
LDPC:	*Low Density Parity Check* - Verificação de Paridade de Baixa Densidade
LLR:	*Log-Likelihood Ratio* – Log-Razão de Verossimilhança
MAP:	*Maximum a Posteriori Probability* - Máxima Probabilidade a Posteriori
PLC:	*Power Line Communications* – Comunicações por Rede Elétrica
PSK:	*Phase Shift-Keying* – Chaveamento por Deslocamento de Fase
QAM:	*Quadrature Amplitude Modulation* - Modulação de Amplitude em Quadratura
SαS:	*Symmetric Alpha-Stable Distribution* – Distribuição Alfa-estável Simétrica
SNR:	*Signal-to-Noise Ratio* - Relação sinal-ruído
SOVA:	*Soft Output Viterbi Algorithm* - Algoritmo de Viterbi com Saída Suave
TCL:	Teorema Central do Limite

SUMÁRIO

1. Introdução

Este trabalho de graduação tem como proposta o estudo e a análise de desempenho de códigos corretores de erros avançados, concentrando-se nos códigos turbo, quando aplicados a sistemas de comunicação sujeitos ao ruído impulsivo.

A especificação do ruído presente no canal de comunicação é um importante requisito para o sucesso do projeto de um sistema de comunicação. O modelo que atribui uma distribuição Gaussiana às amostras de ruído do canal é o mais amplamente utilizado. Neste modelo o canal é denominado AWGN (do inglês a*dditive white Gaussian noise*), em razão do ruído ter três características: é linearmente somado ao sinal transmitido; é branco, pois sua densidade espectral de potência é plana sobre toda a banda de transmissão utilizada e suas amostras têm uma distribuição Gaussiana. Este modelo é bastante razoável em muitos exemplos de aplicações práticas e também é justificado pelo Teorema Central do Limite (TCL) [1].

Porém, em muitos canais de comunicação, são observados ruídos aleatórios que não se caracterizam por uma distribuição de probabilidade Gaussiana. Podem-se enumerar alguns exemplos dessas ocorrências como o ruído devido aos transientes no ligamento/desligamento de equipamentos nos canais da rede elétrica (sistemas PLC, do inglês *Power Line Communication*), ruído de ignição de automóveis, que afetam as comunicações sem fio, ruídos atmosféricos de baixa frequência nas comunicações por satélite e enfim, conforme muitas referências apontam, muitos equipamentos produzidos pelo homem inserem ruídos de características impulsivas nos sistemas de comunicação modernos [2].

Neste cenário, justifica-se a necessidade da verificação do desempenho de códigos corretores de erro, desenvolvidos para decodificar sinais recebidos através de um canal AWGN, visando à construção de códigos mais robustos, ou mesmo à adaptação de códigos existentes, que sejam mais eficazes e eficientes na presença do ruído impulsivo.

Este trabalho considera uma abordagem para modelar e simular computacionalmente o ruído impulsivo, que consiste na sua representação estatística através da distribuição alfa-estável simétrica ($S\alpha S$). Esta distribuição exibe cauda mais "pesada" que a distribuição Gaussiana. Por isso, pode representar melhor ruídos que apresentam ocorrências repentinas de picos pronunciados. Além disso, pela variação do seu expoente característico, a distribuição alfa-estável pode modelar ruídos com maior ou menor nível de impulsividade. Conforme exposto em [3], as distribuições alfa-estáveis têm

sido utilizadas para descrever diferentes fenômenos como flutuações aleatórias de campos gravitacionais, índices do mercado econômico, ecos indesejados em sistemas de radar e na modelagem do tráfego em redes de computadores, além da representação de ruídos em sistemas de comunicação.

No entanto, antes de detalhar as características do ruído alfa-estável, convém descrever os componentes básicos de um sistema de comunicação digital.

Um sistema de comunicação digital moderno pode ser esquematizado no diagrama da Figura 1, na qual estão destacadas as partes deste sistema que são o objeto principal de estudo do presente trabalho, ou seja, a codificação e a decodificação de canal para o controle de erros devido à degradação d

o sinal pelo ruído do canal.

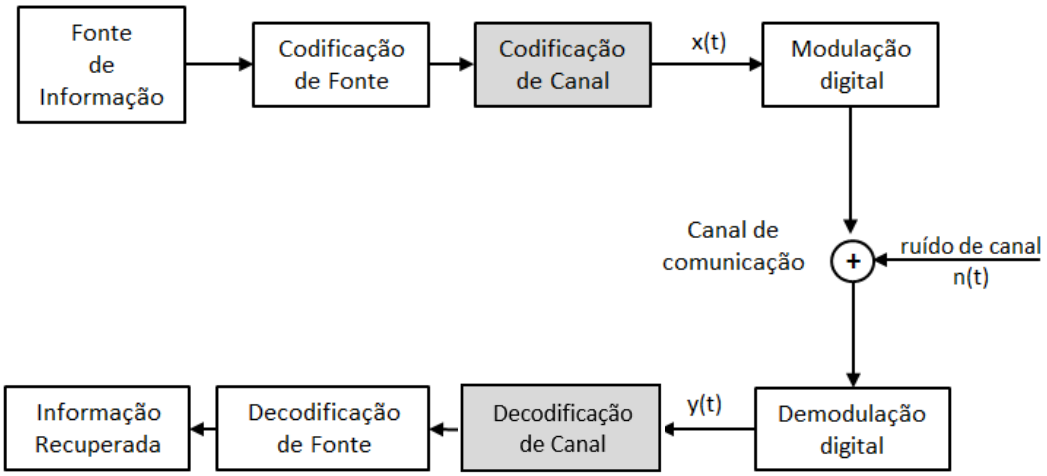

Figura 1 – Diagrama esquemático de um sistema de comunicação

Cada parte do sistema mostrado na Fig. 1, pode ser sucintamente explicada conforme a seguir.

- A fonte de informação consiste na mensagem ou conjunto de mensagens a serem transmitidas ao receptor. A codificação de fonte processa o sinal de informação para que seja reduzida a redundância da mensagem o máximo possível. Fazem parte deste processo a amostragem, quantização, compressão e codificação da mensagem em uma sequência de bits de informação.
- A codificação de canal visa à redução da probabilidade de erro na recepção e, considerando um dado valor de taxa de erro de bit (BER, do inglês *Bit*

Error Rate) requerido, pode-se reduzir a relação sinal-ruído (SNR, do inglês S*ignal-to-Noise Ratio*) necessária, aumentando a eficiência na utilização de potência. As técnicas de codificação de canal consistem genericamente na introdução de bits redundantes na informação a transmitir. Os bits adicionais permitem de um modo geral, a detecção e a correção de erros no sinal recebido. O uso de técnicas de codificação de canal implica na expansão da largura de banda ocupada pelo sinal, pois no mesmo intervalo de tempo, passa-se de um número *k* de símbolos de mensagem para um número mais elevado, *n*, de símbolos codificados.

- A modulação é o processo pelo qual a sequência de informação, em bits ou mapeada em símbolos, irá modular uma forma de onda adequada para ser transmitida pelo canal.
- O canal é o meio usado para transmitir o sinal ao receptor, podendo ser um par de cabos, cabo coaxial, banda de rádio frequência, fibra óptica, etc.
- O receptor irá realizar as operações inversas àquelas efetuadas pelo transmissor, ou seja, a demodulação, decodificação de canal e decodificação de fonte, recuperando a mensagem transmitida.

As técnicas de comunicação digital e a aplicação dos códigos corretores de erros expandiram a capacidade dos sistemas de comunicação, permitindo a recepção de sinais mesmo em condições de baixa SNR.

Claude Elwood Shannon, em clássico artigo científico de 1948 [4], afirma que haveria um limite na quantidade de informação que poderia trafegar por um canal com ruído. Nele estabelece que a maior taxa de transmissão que pode trafegar em um canal AWGN é dada por

$$C = B \,.\, log_2(1 + SNR), \qquad (1)$$

onde *C* é a capacidade do canal em bits por segundo (bps), *B* é a banda disponível em Hertz e SNR é a relação entre as potências do sinal e do ruído, dadas em Watt. Shannon também demonstrou que seria possível reduzir a taxa de erros no canal a um patamar tão pequeno quanto se quisesse, desde que fosse adicionada uma redundância controlada na

informação, ou seja, pela utilização de códigos corretores de erros. Porém, não foi dado por ele o detalhamento de como esses códigos poderiam ser construídos na prática.

Shannon desenvolveu os fundamentos para a comunicação de informação na presença de ruído, afirmando que existem códigos capazes de alcançar probabilidades de erro de decodificação arbitrariamente pequenas, para qualquer taxa de transmissão de dados R menor que a capacidade de canal C. Ele apontou que esses códigos deveriam ser formados por blocos suficientemente longos e mostrou que um conjunto de códigos escolhidos aleatoriamente, somado ao uso da decodificação por máxima verossimilhança, poderia alcançar a capacidade do canal. Uma das consequências do teorema da capacidade de canal mostra que, para uma codificação ótima, é possível aproximar-se de uma probabilidade de erro zero para Eb/N0 (relação da energia de bit de informação pela densidade espectral de potência do ruído) de -1,6 dB. Este valor é conhecido como o Limite de Shannon, porém como esse limite provém de uma taxa de código de valor zero e, por isso, não prática, muitos autores utilizam o limite pragmático de Shannon, cujo valor de Eb/N0 é da ordem de 0,2 dB para canais AWGN binários e taxa de código 1/2 [5].

Até o início dos anos 90, pensava-se que apenas códigos muito longos e impraticáveis eram capazes de operar próximos à capacidade do canal. Códigos LDPC (do inglês, L*ow Density Parity C*heck*)* e códigos turbo demonstraram que um desempenho próximo à capacidade do canal é possível na prática e com complexidade computacional viável.

O objetivo principal deste trabalho é, portanto, estudar a codificação e a decodificação turbo e avaliar o seu desempenho, através de simulação computacional, em um sistema de comunicação cujo canal degrada o sinal pela presença de ruído impulsivo, modelado matematicamente pela distribuição de probabilidade alfa-estável simétrica.

2. Códigos Turbo

Códigos turbo apresentam uma estrutura básica que consiste em dois ou mais códigos convolucionais recursivos sistemáticos concatenados e um entrelaçador. O modelo de código turbo apresentado à comunidade científica por Berrou, Glavieux e Thitimajshima em 1993 [6] é construído, numa descrição sucinta, pela concatenação em paralelo de dois codificadores convolucionais, cujos respectivos decodificadores compartilham informação entre si, de modo iterativo, antes de convergir para uma estimativa da palavra código

transmitida. O nome turbo faz uma analogia ao funcionamento dos motores turbo, pois a informação de saída de um decodificador é realimentada na entrada do outro.

Outros padrões de código turbo foram elaborados e são encontrados na literatura e em aplicações práticas como, por exemplo, aqueles que utilizam a concatenação em série de códigos convolucionais. No entanto, o presente trabalho tem seu foco na exposição teórica e na simulação computacional do desempenho de códigos turbo realizados com dois códigos convolucionais recursivos e sistemáticos concatenados em paralelo.

Cabe primeiramente, uma descrição do código convolucional, pois ele é um código constituinte do código turbo, conforme mencionado no parágrafo anterior.

2.1 Códigos convolucionais

A codificação de canal pode ser dividida em duas grandes classes: os códigos de bloco e os códigos convolucionais. Nos códigos de bloco a informação é agrupada em sequência de k bits e codificada em uma palavra código de n bits após terem sido adicionados $(n - k)$ bits de verificação de paridade. No código de bloco, cada bloco codificado é unicamente determinado pela entrada de dados atual, sendo independente de entradas anteriores. Diferentemente, num código convolucional, as saídas do codificador dependem não somente da entrada atual, mas também de certo número de entradas anteriores, ou seja, o código tem memória.

Um codificador para um código convolucional (n,k,m) consiste de mk registradores de deslocamento (*shift registers*) e n somadores módulo-2. O inteiro m é o parâmetro conhecido como comprimento de restrição (*constraint length*) do código convolucional e está relacionado com a memória do código. Na prática n e k são números inteiros pequenos e m é variado para controlar a redundância.

A Figura 2 mostra um codificador convolucional com $m = 3$ e $n = 2$. A sequência de informação u entra no codificador bit a bit. Cada bit nas saídas codificadas v_1 e v_2 é obtido através da soma módulo-2 da entrada e de alguns estágios do registrador de deslocamento. As saídas são a convolução da entrada u com as sequências geradoras do codificador. As sequências geradoras podem ser escritas na forma polinomial, com os coeficientes do polinômio iguais aos elementos da sequência correspondente. As sequências geradoras são funções de transferência e podem ser obtidas pelas respostas ao impulso do sistema.

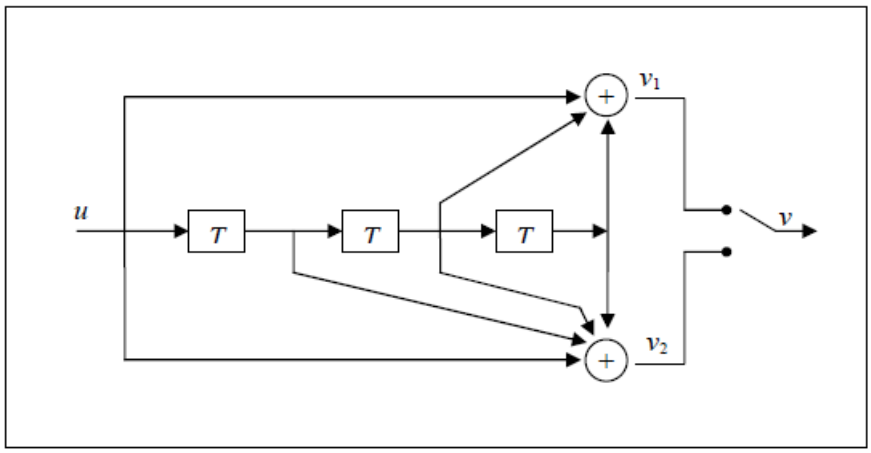

Figura 2 – Código convolucional de taxa ½, k=1, m = 3 e n = 2.

Chamando o conteúdo de cada um dos três estágios do codificador da Figura 2 de X_0, X_1 e X_2 , e os bits de saída codificados de v_1 e v_2 temos

$v_1 = u \oplus X_1 \oplus X_2$

$v_2 = u \oplus X_0 \oplus X_1 \oplus X_2$,

onde $\oplus$ é o símbolo da adição módulo-2 (operação XOR).

O código é executado continuamente em todos os bits enquanto eles vão sendo deslocados pelo registrador de deslocamento. Considerando uma sequência de entrada u = (10111000), para o codificador da Figura 2, as saídas v_1 e v_2 resultantes são mostradas na tabela abaixo.

Tempo	u	X_0	X_1	X_2	v_1	v_2
1	1	0	0	0	1	1
2	0	1	0	0	0	1
3	1	0	1	0	0	0
4	1	1	0	1	0	1
5	1	1	1	0	0	1
6	0	1	1	1	0	1
7	0	0	1	1	0	0
8	0	0	0	1	1	1

Como resumo do resultado obtido no exemplo proposto, abaixo estão descritos a entrada *u* fornecida ao codificador e a correspondente saída *v*, após multiplexação dos bits das sequências v_1 e v_2.

Sequência de entrada (u): 1 0 1 1 1 0 0 0

Sequência de saída (v): 11 01 00 01 01 01 00 11

2.1.1 Resposta ao impulso do codificador convolucional

A resposta ao impulso do codificador convolucional é a sequência de bits gerada na saída quando um único bit "1" se desloca pelos registradores, conforme abaixo:

u	X_0	X_1	X_2	v_1	v_2
1	0	0	0	1	1
0	1	0	0	0	1
0	0	1	0	1	1
0	0	0	1	1	1

Sequência de entrada: 1 0 0 0

Resposta ao impulso: 11 01 11 11

Verifica-se que a resposta ao impulso de v_1 é (1 0 1 1) e a resposta ao impulso de v_2 é (1 1 1 1). Para uma sequência de entrada u = (1 0 1 1 1), a saída resulta da soma linear da resposta ao impulso do codificador deslocada no tempo, conforme é demonstrado abaixo:

Entrada (u)	Saída (v)
1	11 01 11 11
0	00 00 00 00
1	11 01 11 11
1	11 01 11 11
1	11 01 11 11
Adição de Módulo-2:	11 01 00 01 01 01 00 11

A operação resulta na mesma sequência de saída obtida anteriormente, mostrando que os códigos convolucionais são lineares. As sequências v_1 e v_2 são a soma de convolução discreta (módulo-2) da sequência de entrada com suas respectivas respostas ao impulso, portanto vem daí o nome convolucional atribuído ao código.

2.1.2 Representação pelo polinômio gerador

No exemplo da Figura 2 os polinômios geradores do código podem ser escritos como

$$g_1(x) = 1 + x^2 + x^3$$
$$g_2(x) = 1 + x + x^2 + x^3 .$$

A sequência de saída é então $c(x) = d(x)\, g_1(x)$ multiplexada com $d(x)\, g_2(x)$. As sequências geradoras são $g_1 = (1011)$ e $g_2 = (1111)$ ou $g_1 = 13$ e $g_2 = 17$, em octal.

Os polinômios geradores podem ser também representados por

$$G^1(D) = 1 + D^2 + D^3$$
$$G^2(D) = 1 + D + D^2 + D^3,$$

onde D pode ser interpretado como um operador de atraso e as potências de D denotam o número de unidades de tempo que um bit é atrasado em relação ao bit inicial da sequência.

Para a mesma sequência de entrada u = (10111), ou seja, $u(D) = 1 + D^2 + D^3 + D^4$, teremos as seguintes sequências de saída:

$$v_1(D) = (1 + D^2 + D^3 + D^4)(1 + D^2 + D^3) = 1 + D^7 \rightarrow (10000001)$$

$$v_2(D) = (1 + D^2 + D^3 + D^4)(1 + D + D^2 + D^3) = 1 + D + D^3 + D^4 + D^5 + D^7 \rightarrow (11011101).$$

Para uma única entrada e sequência de informação de tamanho L, a sequência codificada terá tamanho n(L + m). Dessa forma, a taxa de codificação será dada por

$$R = \frac{L}{n(L+m)} . \quad (2)$$

Se L >> m, pode-se considerar $\frac{L}{(L+m)} \approx 1$ e $R = \frac{1}{n}$. (3)

2.1.3 Diagrama de Estados

O estado de um codificador convolucional (n,1,m) é definido como o conteúdo dos elementos de memória. Conhecendo o estado presente e a próxima entrada, pode-se determinar o próximo estado e então, a saída. A transição entre estados é governada pelo bit de entrada (0 ou 1). Um codificador possui 2^{υ} estados, onde υ é a ordem de sua memória, que consiste no número de elementos registradores de deslocamento (*shift registers*). Para o código (2,1,3) da Figura 2, o diagrama correspondente é mostrado na Figura 3. O código possui oito (2^3) estados (S0 a S7). Cada nó do diagrama representa um estado e as linhas representam as possíveis transições de estado. Cada linha está rotulada com a nomenclatura x/y_1y_2, sendo x o bit de entrada que produziu a transição e y_1y_2 a saída gerada.

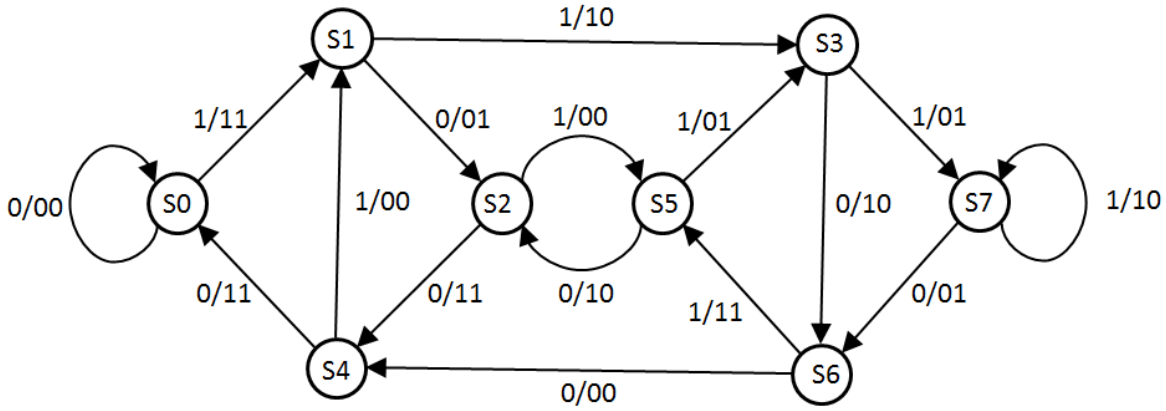

Figura 3 – Diagrama de estados para o código convolucional (2,1,3) da Fig. 2.

Continuando com o mesmo exemplo das seções anteriores, para uma entrada u = (10111000), através do diagrama de estados da Figura 3, pode-se verificar que a saída será v = (11 01 00 01 01 01 00 11).

2.1.4 Diagrama de Treliça

O diagrama de estados descreve completamente a relação entre entrada-saída de um codificador convolucional, porém não mostra a evolução temporal das transições de estado. Para esta finalidade é empregado o diagrama de treliça. A Figura 4 mostra um diagrama de treliça para um codificador convolucional (2,1,2) com geradores

$G_1(D) = 1 + D^2$ e $G_2(D) = 1 + D + D^2$. À esquerda cada estado é representado para o tempo t e à direita uma cópia de cada estado é representada para o tempo (t +1).

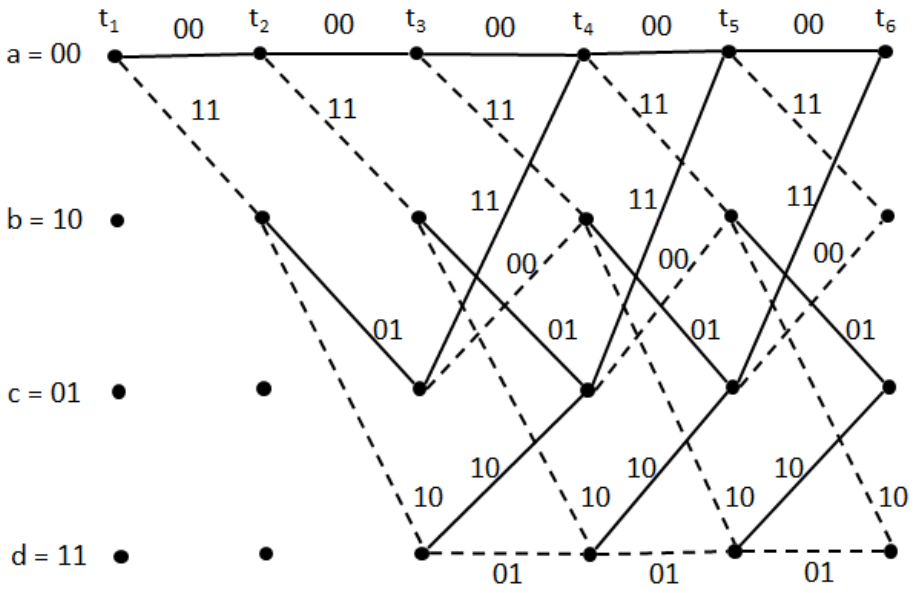

Figura 4 – Diagrama de treliça para o código convolucional (2,1,2) com geradores G1(D) = 1 + D² e G2(D) = 1 + D + D².

As linhas cheias denotam as saídas geradas quando o bit de entrada é "0" e as linhas tracejadas denotam as saídas geradas quando o bit de entrada é "1". Os números que aparecem em cada ramo da treliça correspondem aos bits de saída do codificador. A estrutura da treliça é utilizada para a decodificação da sequência codificada através da obtenção do caminho mais provável que foi percorrido enquanto o código foi gerado. O diagrama de treliça é especialmente útil para a decodificação probabilística.

2.1.5 Mínima Distância Livre (d_{free})

A mínima distância livre (d_{free}) de um código convolucional é a menor distância de Hamming entre quaisquer duas palavras código. Em geral, quanto maior a distância livre, maior será a capacidade do código para detectar e corrigir erros ocorridos na transmissão da informação. Em virtude dos códigos convolucionais serem lineares, toda palavra código tem o mesmo número de outras palavras código que diferem dela por *d* localizações de bit, para qualquer valor de *d* [7]. A distância livre do código convolucional é também o menor peso de Hamming das palavras código terminadas geradas por sequências que não sejam

constituídas apenas por zeros. Define-se o peso de Hamming de uma palavra código como o número de "1s" contidos nela. Palavras código terminadas são aquelas formadas pelo caminho no diagrama de treliça que inicia e termina no estado zero. Sendo assim, a mínima distância livre do código convolucional pode ser definida como o menor peso de Hamming dentre todos os caminhos da treliça que iniciam e terminam no estado zero. Pode-se aumentar a mínima distância livre (d_{free}) de um código convolucional aumentando-se a sua ordem de memória (υ). Contudo, o aumento de uma unidade na ordem de memória, aumenta a complexidade na decodificação em 2^{υ}. Por isso, há um custo na escolha entre desempenho e complexidade a ser considerado.

2.1.6 Código Convolucional Recursivo

Num código convolucional recursivo, os bits de paridade são gerados por malhas de realimentação. A Figura 5 mostra um exemplo de código convolucional recursivo, onde uma das saídas é realimentada à entrada do registrador de deslocamento. Um código é sistemático quando as primeiras *k* saídas são réplicas exatas das k entradas, ou seja, o conteúdo da informação de entrada aparece sem modificação no conteúdo da palavra código. No exemplo da Figura 5, a entrada $u^{(1)}$ está diretamente mapeada na saída $c^{(1)}$ e a saída $c^{(2)}$ está sendo realimentada na entrada do registrador de deslocamento, por isso esse é um exemplo de código convolucional sistemático recursivo (*RSC - Recursive Systematic Convolutional*). Os códigos RSC são empregados como códigos constituintes dos códigos turbo.

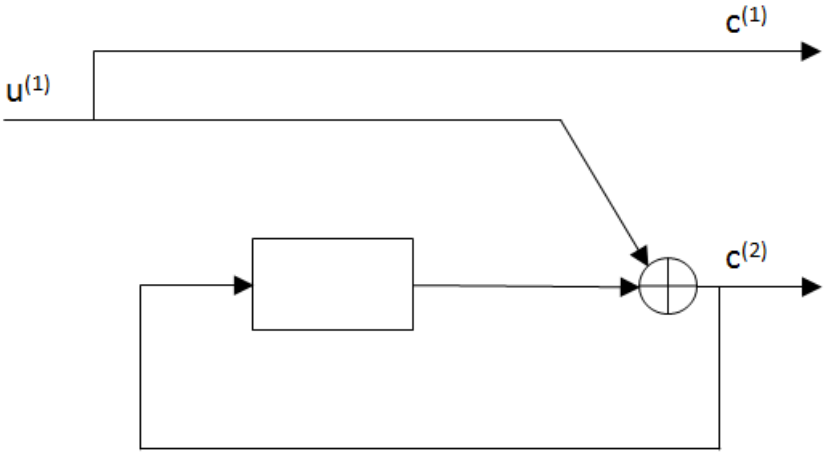

Figura 5 – Código Convolucional Sistemático Recursivo

O codificador da Figura 5 possui um único elemento registrador, o qual será denominado como $s^{(1)}$. O elemento registrador $s^{(1)}$ pode assumir dois estados, ou seja, S0 para $s^{(1)} = 0$ e S1 para $s^{(1)} = 1$.

Considerando que os bits de saída e de entrada podem assumir diferentes valores em cada intervalo de tempo, será usada a nomenclatura $u_t^{(1)}$ para o valor do bit de entrada, $c_t^{(1)}$ para o valor do bit da saída 1, $c_t^{(2)}$ para o valor do bit da saída 2 e $s_t^{(1)}$ para o valor do elemento registrador.

O bit de saída $c_t^{(1)}$ é $u_t^{(1)}$ e o bit de saída $c_t^{(2)}$ é o bit de paridade formado pela adição módulo-2 do bit de entrada $u_t^{(1)}$ e o valor do elemento registrador de deslocamento $s_t^{(1)}$, podendo-se escrever as equações

$c_t^{(1)} = u_t^{(1)}$ e $c_t^{(2)} = u_t^{(1)} \oplus s_t^{(1)} = u_t^{(1)} \oplus c_{t-1}^{(2)}$ para representar este codificador.

A saída no tempo *t* é função da entrada no tempo *t* e da saída no tempo (*t – 1*), que por sua vez é função da entrada no tempo (*t – 1*) e da saída no tempo (*t – 2*), e assim por diante, ou seja, $c_t^{(2)}$ é função de todas as entradas anteriores. Por esta razão os códigos convolucionais recursivos tem resposta ao impulso infinita *(IIR – Infinite Impulsive Response*). Pode-se encontrar o polinômio gerador em *D* para este código, resolvendo para $c^{(2)}$:

$$c^{(2)} = u^{(1)} + c^{(2)}D \pmod 2,$$
$$c^{(2)}(1 + D) = u^{(1)} \pmod 2,$$

$$c^{(2)} = \frac{u^{(1)}}{1 + D},$$

que resulta no polinômio gerador para $c^{(2)}$:

$$g_2(D) = \frac{1}{1 + D} \ .$$

O gerador para $c^{(1)}$ é simplesmente $g_1(D) = 1$ e então temos a matriz geradora, conforme abaixo,

$$G = \left[1 \quad \frac{1}{1+D}\right].$$

2.1.7 Decodificação de códigos convolucionais

A codificação de uma mensagem só tem propósito se a mensagem original puder ser recuperada a partir da informação codificada. Assim, a tarefa do decodificador é estimar a mensagem original baseando-se no sinal recebido que, para o código convolucional, consiste em encontrar o caminho da treliça que gerou a palavra código enviada. É preciso escolher um dentre os vários caminhos possíveis.

Para decodificar os códigos convolucionais, um dos métodos mais usados é conhecido como algoritmo de Viterbi, que realiza a decodificação por máxima verossimilhança [8].

No receptor, temos a sequência de sinais recebidos correspondendo aos bits codificados enviados pelo transmissor. Se os sinais recebidos forem quantizados antes de serem decodificados, o processo de decodificação será denominado decodificação por decisão abrupta (*hard decision decoding*). Se os sinais recebidos forem diretamente decodificados, sem quantização, o processo de decodificação será denominado decodificação por decisão suave (*soft decision decoding*). A decodificação de Viterbi pode ser usada para ambos os casos.

Quando as palavras código são equiprováveis, o decodificador que escolhe $\hat{\mathbf{c}}$ se

$$P(r\backslash\hat{c}) = max_{ci}\, P(r\backslash c_i), \tag{4}$$

*on*de $\boldsymbol{r}$ é a sequência recebida, $\hat{\boldsymbol{c}}$ é a palavra código transmitida e c_i é uma das possíveis sequências transmitidas, é chamado de decodificador de máxima verossimilhança *(maximum likelihood decoder).*

O algoritmo de Viterbi opera essencialmente obtendo a distância mínima de Hamming entre o sinal recebido e as possíveis sequências transmitidas, porém favorecendo-se da estrutura da treliça para reduzir o esforço computacional. Como mostra o exemplo da Figura 4, cada um dos quatro estados (*a, b, c* e *d*) pode ser alcançado por apenas dois estados. Assim, apenas toma-se o caminho que mais se aproxima da sequência recebida para cada estado. O caminho tomado é chamado de sobrevivente para aquele

determinado estado. A cada ramo do caminho sobrevivente é atribuída uma métrica, que equivale à mínima distância de Hamming da sequência recebida *r* para o ramo correspondente. A soma das métricas dos ramos resulta na métrica do caminho e, a sequência *r* é decodificada através do caminho sobrevivente com a menor métrica.

2.2 Codificador turbo

A Figura 6 mostra o diagrama de blocos de um codificador de um sistema de código turbo. O primeiro codificador codifica diretamente a sequência de bits **u**, formando a sequência de bits de paridade $p^{(1)}$. O segundo codifica a mesma sequência de bits **u**, porém permutada pelo *Entrelaçador* (π), gerando a sequência de bits de paridade $p^{(2)}$. O código turbo é sistemático, portanto a palavra código resulta, após a multiplexação das saídas, em $v = \{u_1, p_1^{(1)}, p_1^{(2)}, \ldots, u_K, p_K^{(1)}, p_K^{(2)}\}$.

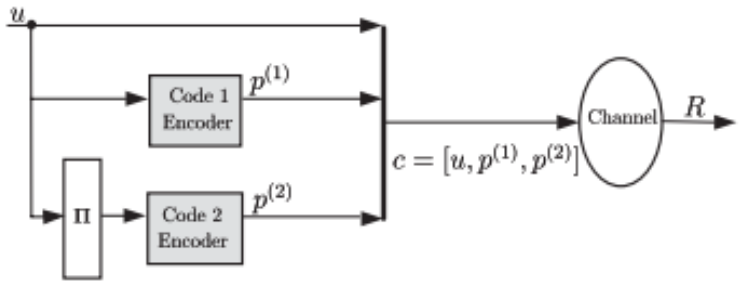

Figura 6 – Diagrama de blocos de um sistema de código turbo com concatenação paralela. Figura extraída da referência [9].

O codificador proposto em [6] é formado por dois componentes constituintes – códigos convolucionais sistemáticos recursivos – concatenados em paralelo, ambos com taxa de código 1/2 e com matriz geradora

$$G = \left[1 \ \frac{1+D^4}{1+D+D^2+D^3+D^4}\right].$$

Como mostra a Figura 7, os componentes em paralelo produzem um código de taxa 1/3, que após o processo de *puncionamento*, que será detalhado na seção 2.2.2, resulta em um código de taxa 1/2.

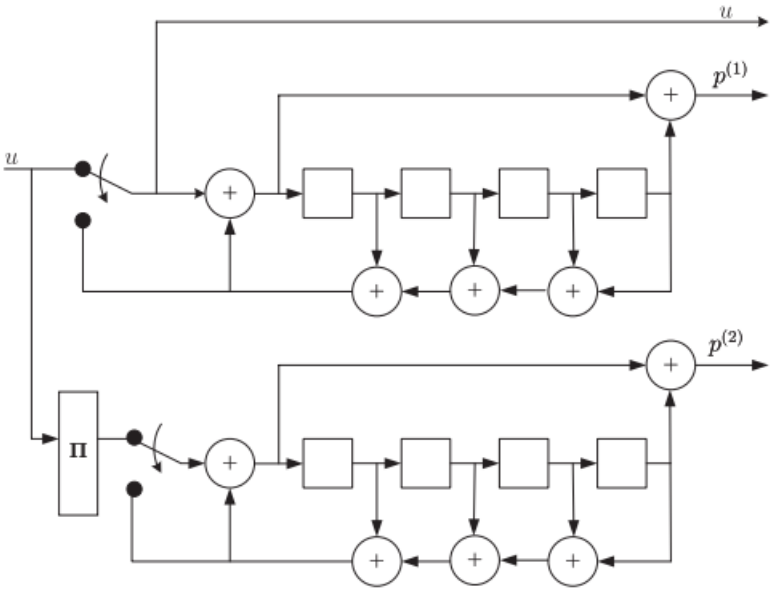

Figura 7 – O codificador turbo. Figura extraída da referência [9].

Os códigos convolucionais recursivos têm realimentação e resposta ao impulso infinita, assim sua função de transferência é uma função racional $G(x) = \frac{h(x)}{g(x)}$. Esta característica pode explicar a razão dos sistemas com codificadores *RSC* terem melhor desempenho para baixos valores de $E_b/N_{0,}$ comparados com códigos não recursivos, uma vez que os bits de informação codificados são continuamente realimentados na entrada do codificador. Benedetto et al [10] mostra que códigos convolucionais recursivos produzem palavras código de alto peso na sua saída, comparados aos códigos convolucionais não recursivos, mesmo quando a mensagem na entrada é formada por uma sequência de baixo peso.

2.2.1 Entrelaçador

Um entrelaçador é representado por uma sequência de permutação $\Pi = [\pi_{1,} \pi_{2,\ ...,} \pi_n]$, onde a sequência $[\pi_{1,} \pi_{2,\ ...,} \pi_n]$ é uma permutação de inteiros de 1 a n. Não há razão para especificar um comprimento de mensagem para um codificador convolucional, pois o codificador pode gerar o código num fluxo de bits, indefinidamente. Contudo, para um código turbo, a existência do entrelaçador necessariamente pré-define um número de bits a ser permutado, ou seja, o entreleçador tem um comprimento.

Uma das funções do entrelaçador, na recepção, é distribuir o erro residual sobre todo o bloco de bits recebidos.

A permutação garante que os bits de paridade produzidos pelo segundo codificador serão completamente diferentes dos produzidos pelo primeiro. Se a sequência 1 de bits de paridade for uma sequência de baixo peso de Hamming, a sequência 2 deverá ser de alto peso, evitando palavras código de baixo peso na codificação turbo [9].

Para alcançar um desempenho próximo ao limite de Shannon, o tamanho do bloco de informação *N*, que corresponde ao tamanho do entrelaçador deve ser bem grande, usualmente da ordem de milhares de bits.

2.2.2 Puncionamento

O procedimento de puncionamento (do inglês, *puncturing)*, realizado antes da multiplexação, consiste na retirada de parte dos bits de paridade dos códigos constituintes. O procedimento é geralmente representado por uma matriz tal como

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

A primeira coluna indica quais bits são saídas nos instantes pares e a segunda coluna indica quais bits são saídas nos instantes ímpares. Por exemplo, considerando uma sequência de entrada de peso-2 **u** = (1000010000000000) e o codificador da Figura 7, será produzida uma sequência de paridade de baixo peso $\mathbf{p}^{(1)}$ = (1100110000000000). Considerando um padrão de permutação de tamanho 16 para o entrelaçador

$$\Pi_{16} = [0,8,15,9,4,7,11,5,1,3,14,6,13,12,10,2],$$

irá resultar na entrada do segundo codificador a sequência **u'** = (1000000100000000), que produz a sequência de paridade de alto peso $\mathbf{p}^{(2)}$ = (1100101111000110).

Quando os três bits são multiplexados juntos, o fluxo de bits da palavra código será

$$\mathbf{c} = (1,1,1,\ 0,1,1,\ 0,0,0,\ 0,0,0,\ 0,1,1,\ 1,1,0,\ ....)$$

e a sequência após puncionamento, conforme matriz P, será

$$\mathbf{c} = (1,1,\ 0,1,\ 0,0,\ 0,0,\ 0,1,\ 1,0,\ ....).$$

2.2.3 Terminação do codificador

Uma treliça de um codificador convolucional pode ser terminada, ou seja, pode retornar ao estado zero através da adição de alguns bits de cauda ao final da mensagem de informação que obriguem o percurso na treliça a regressar ao estado nulo inicial.

Ambos os codificadores constituintes de um código turbo podem ser terminados desta maneira, porém como a mensagem de entrada de cada codificador não é a mesma, a sequência de bits de cauda para terminar cada um também não será a mesma.

2.3 Decodificador turbo

Os dados multiplexados e codificados ***v*** são modulados e transmitidos através do canal, cuja saída é o vetor recebido ***r***. O vetor recebido ***r*** é demodulado e demultiplexado nos vetores $\boldsymbol{r^{(u)}}$ (correspondendo à mensagem ***u*** transmitida), $\boldsymbol{r^{(p1)}}$ (correspondendo aos bits de paridade $p^{(1)}$ do codificador 1) e $\boldsymbol{r^{(p2)}}$ (correspondendo aos bits de paridade $p^{(2)}$ do codificador 2). A operação geral realizada pelo decodificador turbo está resumida na Figura 8.

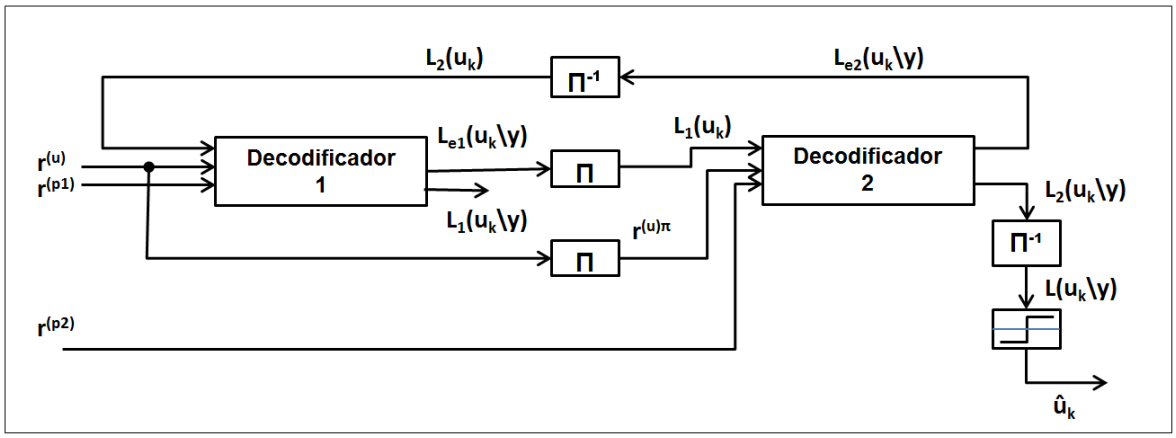

Figura 8 – Diagrama de Blocos do Decodificador Turbo

A decodificação turbo utiliza o processo de decodificação suave. Esse processo também pode ser empregado para a decodificação de códigos convolucionais e empregao a decodificação de máxima probabilidade a posteriori, ou *MAP* (*Maximum a posteriori*), através do algoritmo *BCJR*, proposto por Bahl et al [11], em 1974. A principal diferença dos decodificadores *BCJR*, quando usados em decodificadores turbo, é a fonte de sua informação *a priori*. Para o código convolucional isolado a informação *a priori* vem do

conhecimento da fonte, enquanto que, para o código turbo, cada decodificador utiliza como informação *a priori* a saída do outro decodificador. Esta informação extra que é passada entre os decodificadores é chamada de informação *extrínseca* ou informação de realimentação. Outra diferença é que o algoritmo *BCJR* é usado várias vezes, de modo iterativo, ou seja, a informação extrínseca é atualizada e passada para frente e para trás entre os decodificadores por muitas iterações. O processo continua até que o decodificador determina que o processo convergiu, ou até que um número máximo de iterações tenha sido alcançado, ou por outro critério de parada.

Nos parágrafos a seguir será feita uma breve descrição do algoritmo *BCJR*, comentando-se as etapas e cálculos envolvidos. Será considerado que o sistema de comunicação usa uma modulação BPSK, em que o bit "1" é mapeado para o símbolo "+1" e o bit "0", para o símbolo "-1".

O bit de informação $\boldsymbol{u_k}$, que dá origem, no instante $\boldsymbol{k}$, ao símbolo codificado $\boldsymbol{x_k}$, pode tomar os valores -1 ou +1 com uma probabilidade de ocorrência a priori $\boldsymbol{P(u_k)}$. Associada a esta probabilidade define-se a quantidade

$$L(u_k) = ln\frac{P(u_k = +1)}{P(u_k = -1)}, \tag{5}$$

à qual se dá o nome, em inglês, de *LLR* (*log-likelihood ratio*). Se os bits $\boldsymbol{u_k} = \pm 1$ forem equiprováveis esta *LLR* a priori é nula. A partir da sequência recebida $\boldsymbol{y}$ o algoritmo *BCJR* tentará estimar a sequência de bits originais u_k. Para isso o algoritmo calcula a *LLR* a posteriori $L(u_k\backslash y)$, um valor real definido pela expressão

$$\mathrm{L}(\mathrm{u_k}\backslash \mathrm{y}) = \ln\frac{\mathrm{P}(\mathrm{u_k} = +1\backslash \mathrm{y})}{\mathrm{P}(\mathrm{u_k} = -1\backslash \mathrm{y})}. \tag{6}$$

O sinal, positivo ou negativo, da *LLR*, indica que o bit enviado foi +1 ou -1, respectivamente, e o seu valor absoluto traduz a maior ou menor confiança na decisão por um dos dois valores, isto é, quanto mais afastado o valor de $L(u_k\backslash y)$ estiver do limiar de decisão nulo mais confiança haverá na estimativa do bit. As probabilidades contidas na Eq. (6) são probabilidades a posteriori, pois são calculadas após o conhecimento da sequência recebida $\boldsymbol{y}$. Em outras palavras, têm-se as probabilidades de no instante $\boldsymbol{k}$ o bit de entrada do codificador ter sido $u_k = +1$ ou $u_k = -1$, dado que conhecemos toda a sequência recebida.

A razão *LLR* a posteriori $L(u_k|y)$ produzida na saída de cada decodificador pode ser decomposta numa soma de três parcelas:

$$L(u_k|\mathbf{y}) = L(u_k) + L_c y_{k1} + L_e(u_k)\,. \tag{7}$$

A parcela $L_c y_{k1}$ é proveniente do canal (sequência recebida), $L(u_k)$ vem do conhecimento a priori da ocorrência de bits e $L_e(u_k)$ corresponde à informação extrínseca produzida durante a decodificação e só depende dos bits de paridade da palavra código. Esta informação extrínseca é uma estimativa da *LLR a priori* $L(u_k)$ e pode ser obtida, isolando-a na equação (7):

$$L_e(u_k) = L(u_k|\mathbf{y}) - L(u_k) - L_c y_{k1} \quad . \tag{8}$$

$L_e(u_k)$ é um valor mais preciso da *LLR a priori* e deve substituir o valor anterior de $L(u_k)$. Se o procedimento for repetido de modo iterativo fornecendo a outro decodificador $L_c y_{k1}$ e a nova $L(u_k) = L_e(u_k)$, espera-se obter uma $L(u_k|y)$ mais rigorosa na sua saída.

A Eq. (7) é a base da decodificação iterativa. Na primeira iteração a *LLR a priori* $L(u_k)$ é nula se considerarmos que os bits de entrada são equiprováveis. A informação extrínseca $L_e(uk)$, que cada decodificador fornece, será usada para atualizar $L(u_k)$ de iteração para iteração e de um decodificador para o outro. Desta maneira o decodificador turbo ganha progressivamente mais confiança nas decisões abruptas ±1 que o decisor forçosamente terá de tomar no fim do processo iterativo. A Figura 8 mostra um diagrama de blocos simplificado de um decodificador turbo que ilustra os procedimentos iterativos.

A decodificação iterativa prossegue do seguinte modo:

- Na primeira iteração assumimos que $L(u_k) = 0$. O decodificador 1 fornece então a informação extrínseca $L_{e1}(u_k|y)$ sobre o bit sistemático, ou de mensagem, que obteve do primeiro bit de paridade.
- Após entrelaçamento apropriado a informação extrínseca $L_{e1}(u_k|y)$ do decodificador 1, calculada através da Eq. (8), é entregue ao decodificador 2 como $L_1(u_k)$, que é um "palpite" melhor e mais atual sobre $L(u_k)$. Depois, o decodificador 2 fornece $L_{e2}(u_k|y)$, que é a sua própria informação extrínseca sobre o bit sistemático, mas agora baseada no outro bit de paridade. Após desentrelaçamento conveniente esta informação é entregue ao decodificador 1 como $L_2(u_k)$, a qual é um palpite ainda mais bem elaborado sobre $L(u_k)$. Começará então uma nova iteração.

- Após um número pré-determinado de iterações, ou após ser atingido um determinado critério de parada, a *LLR* $L_2(u_k\backslash y)$ na saída do decodificador 2 é desentrelaçada e entregue como $L(u_k\backslash y)$ ao dispositivo de decisão abrupta (do inglês *hard decision*) o qual, por sua vez, estima o bit de informação de acordo exclusivamente com o sinal, positivo ou negativo, da *LLR* desentrelaçada.

2.4 Cálculo da LLR A Posteriori

Nesta seção são descritos os cálculos realizados pelo algoritmo *BCJR- MAP* para obter as probabilidades condicionais *a posteriori*. Para isto, é conveniente utilizar o diagrama de treliça. Será considerada a aplicação de um codificador convolucional de taxa 1/2, com memória m = 2 e, portanto, quatro estados S = {0, 1, 2, 3}, e um diagrama de treliça entre dois instantes de tempo consecutivos como o apresentado na Figura 9.

Por convenção será considerado que um bit de mensagem -1 na entrada do codificador produz um ramo de traço contínuo e um bit +1 produz um ramo tracejado. Cada ramo está rotulado com a correspondente saída de dois bits x_k, em que 0 e 1 correspondem a -1 e +1, respectivamente.

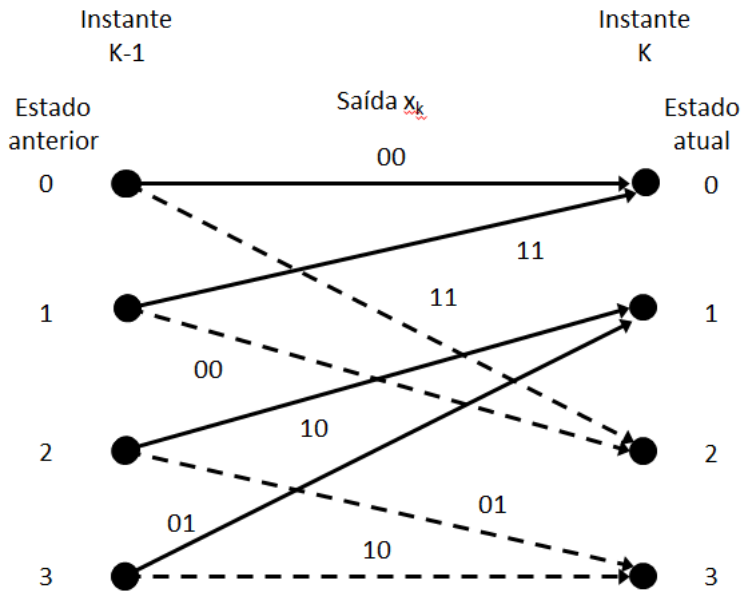

Figura 9 – Diagrama de treliça

Supondo que se esteja no instante *k*, o estado correspondente é $S_k = s$, o estado anterior é $S_{k-1} = s'$ e o símbolo recebido no decodificador é y_k. Antes deste instante já

tinham sido recebidos *k-1* símbolos e depois irão ser recebidos (*N* - *k*) símbolos. Ou seja, a sequência completa ***y*** pode ser dividida em três subsequências, uma representando o passado, outra o presente e outra o futuro:

$$\mathbf{y} = \underbrace{y_1 y_2 \cdots y_{k-1}}_{\mathbf{y}_{<k}} y_k \underbrace{y_{k+1} \cdots y_N}_{\mathbf{y}_{>k}} = \mathbf{y}_{<k}\, \mathbf{y}_k\, \mathbf{y}_{>k} \quad . \tag{9}$$

A *LLR a posteriori* $L(u_k \backslash y)$ é dada pela expressão:

$$L(u_k \,|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)} \quad , \tag{10}$$

sendo que $P(s', s, y)$ representa a probabilidade conjunta de que no instante (*k* - *1*) se está no estado *s'*, no instante corrente, *k*, se está no estado s e de que a sequência de *N* bits recebida é ***y***. Na operação realizada no numerador, R_1 significa que o somatório se estende às transições entre estados *s'* e *s* provocadas por um bit de mensagem u_k = +1 (ou seja, ramos tracejados). Da mesma maneira, no denominador, R_0 designa os outros ramos, os que são originados por um bit de mensagem u_k = -1. As variáveis α, γ e β são probabilidades definidas a seguir.

2.4.1 **Cálculo da probabilidade conjunta** *P(s', s, **y**)*

A probabilidade conjunta $P(s', s, \mathbf{y})$ pode ser calculada pelo produto de três probabilidades:

$$P(s', s, \mathbf{y}) = \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s), \tag{11}$$

sendo que as parcelas à direita da igualdade podem ser definidas como:

$$\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k}) \tag{12}$$

$$\gamma_k(s', s) = P(\mathbf{y}_k, s \backslash s') \tag{13}$$

$$\beta_k(s) = P(\mathbf{y}_{>k} \backslash s) \tag{14}$$

No instante *k* as probabilidades *α*, *γ* e *β* estão associadas ao passado, ao presente e ao futuro da sequência *y*, respectivamente. A seguir verifica-se como se dá o cálculo destas probabilidades.

2.4.1.1 Cálculo de γ

A probabilidade $\gamma_k(s', s) = P(\mathbf{y_k}, s \backslash s')$ é a probabilidade condicional de, no instante *k*, o símbolo recebido ser y_k e o estado atual ser $S_k = s$, sabendo-se que o estado de onde se proveio foi S_{k-1}. É dada pelo produto de probabilidades

$$\gamma_k(s', s) = P(y_k \backslash x_k) P(u_k) \tag{15}$$

No caso especial de canais AWGN $\gamma_k(s', s)$ é dada por

$$\gamma_k(s', s) = C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2} \sum_{l=1}^{n} x_{kl} y_{kl} \right), \tag{16}$$

onde C_k representa uma quantidade que, surgindo no numerador e no denominador da Eq. (11), vai desaparecer ao calcular-se a *LLR* condicional $L(u_k \backslash y)$ e, portanto, é cancelado. Na segunda exponencial a quantidade L_c é a medida de confiabilidade do canal e é igual a

$$\mathrm{Lc} = 4\mathrm{a}\frac{E_c}{N_0} = 4\mathrm{aR_c}\frac{E_b}{N_0} \ . \tag{17}$$

Onde $N_0/2$ é a densidade espectral de potência bilateral do ruído, E_c e E_b são as energias transmitidas por bit codificado e bit de mensagem, respectivamente, R_c é a taxa do código e *a* é a amplitude de *fading*. Se no canal não existir *fading*, *a = 1*.

2.4.1.2 Cálculo recursivo de α e β

As probabilidades α e β são calculadas recursivamente. As respectivas fórmulas recursivas são as seguintes:

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s)\alpha_{k-1}(s') \qquad \text{Condições iniciais: } \alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{18}$$

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s', s)\beta_k(s) \qquad \text{Condições iniciais: } \beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{19}$$

Algumas observações importantes precisam ser consideradas:

• Em ambos os casos, necessita-se da mesma quantidade $\gamma_k(s', s)$, que terá de ser calculada primeiro.

• No caso de $\alpha_k(s)$ os somatórios são efetuados para todos os estados anteriores dos quais saem ramos que convergem no estado s, enquanto que no caso de $\beta_{k-1}(s')$ os somatórios são efetuados para todos os estados seguintes s que se atingem do estado s'.

• A probabilidade α vai sendo calculada à medida que se for recebendo a sequência y. Ou seja, o cálculo de α vai do início para o fim da treliça (sentido "*forward*").

• A probabilidade β só pode ser calculada depois de ser recebida toda a sequência y. Ou seja, o cálculo de β vai do fim para o princípio (sentido "*backward*").

• Na treliça os valores de α e β estão associados aos estados do codificador e os valores de γ estão associados aos ramos ou transições entre estados.

• Os valores iniciais $\alpha_0(s)$ e $\beta_N(s)$ pressupõem que o percurso na treliça começa e termina no estado nulo (treliça terminada).

A partir das considerações mencionadas acima fica evidente a razão pela qual o algoritmo *BCJR* também é conhecido como "*forward-backward algorithm*".

2.4.1.3 Normalização

Para dirimir os problemas de instabilidade numérica associados ao algoritmo *BCJR* e evitar situações de "*underflow*" ou "*overflow*" as probabilidades devem ser previamente normalizadas. Nesse sentido, em vez de na Eq. (11) usar α e β calculadas diretamente das equações recursivas (18) e (19), essas probabilidades devem ser previamente normalizadas pela soma de todos os α e β em cada instante. O mesmo se aplica à probabilidade conjunta, $P(s', s, y)$, como segue. Em cada instante de tempo k definem-se as variáveis auxiliares não normalizadas α' e β':

$$\alpha'_k(s) = \sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s) \tag{20}$$

$$\beta'_{k-1}(s') = \sum_{s} \beta_k(s)\gamma_k(s',s) \tag{21}$$

Depois de todos os M valores de α' e β' terem sido calculados, faz-se a soma $\sum_s \alpha'_k(s)$ e $\sum_{s'} \beta'_{k-1}(s)$. Depois se normaliza α e β pelas somas:

$$\alpha_k(s) = \frac{\alpha'_k(s)}{\sum_s \alpha'_k(s)} \tag{22}$$

$$\beta_{k-1}(s') = \frac{\beta'_{k-1}(s')}{\sum_{s'} \beta'_{k-1}(s')} \tag{23}$$

Do mesmo modo, depois de todos os 2M produtos $\alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)$,em todos os ramos da treliça terem sido calculados no instante *k*, a sua soma,

$$P_{norm}(s',s,\mathbf{y}) = \frac{P(s',s,\mathbf{y})}{\Sigma_{Pk}} \tag{24}$$

irá normalizar P(s′, s, y):

$$\begin{aligned}\Sigma_{Pk} &= \sum_{R_o,R_1} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s) = \\ &= \sum_{R_o} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s) + \sum_{R_1} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)\end{aligned} \tag{25}$$

.

Assim se garante que as somas de todos os α, β e *Pnorm (s', s, y)* são sempre iguais a 1 em cada instante *k*. Nenhuma destas somas de normalização afeta o *LLR L(uk\y)* final dado que todas aparecem no numerador e no denominador:

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s',s,\mathbf{y})}{\sum_{R_0} P(s',s,\mathbf{y})} = \ln \frac{\sum_{R_1} P_{norm}(s',s,\mathbf{y})}{\sum_{R_0} P_{norm}(s',s,\mathbf{y})} \tag{26}$$

2.4.1.4 Cálculo de α e β usando a treliça

A Figura 10 apresenta a treliça da Figura 9, porém agora com outros rótulos a serem explicados a seguir.

• em cada ramo foi colocado um rótulo com o valor de $\gamma_k(s', s)$ calculado de acordo com a Eq. (16).

• em cada nó de estado foi colocado o valor $\alpha_k(s)$ calculado de acordo com a Eq. (18) ou (22) a partir das condições iniciais $\alpha_0(s)$.

• em cada nó de estado e por baixo do valor de $\alpha_k(s)$ foi colocado o valor de $\beta_k(s)$ calculado de acordo com a Eq. (19) ou (23) a partir das condições iniciais $\beta_N(s)$.

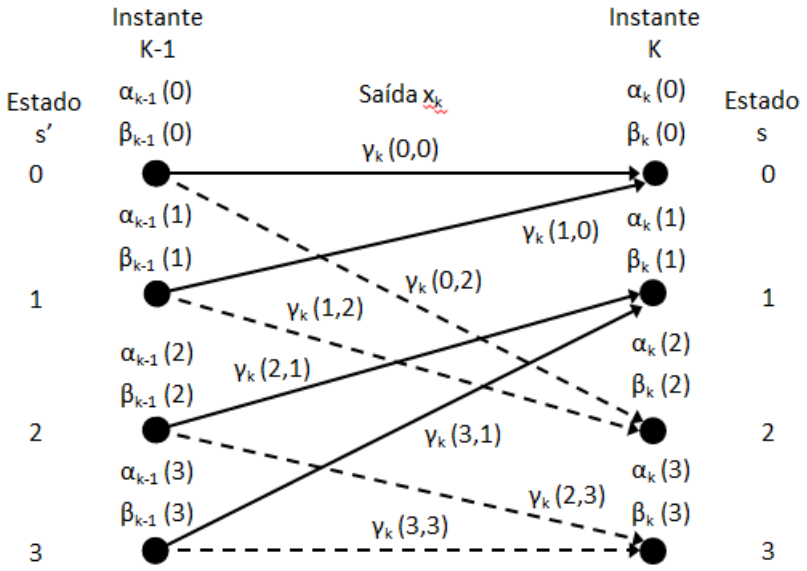

Figura 10 – Treliça com os valores de α, β e γ.

2.4.1.5 Cálculo de α

Supondo que se conhece o valor de $\alpha_{k-1}(s')$, a probabilidade $\alpha'_k(s)$ (sem normalização) é obtida somando os produtos de $\alpha_{k-1}(s')$ e $\gamma_k(s',s)$ dos ramos que convergirem em s. Por exemplo, e olhando para a treliça da Figura 10, vê-se que no instante k chegam dois ramos ao estado $s = 2$, um que vem do estado 0 e outro que vem do estado 1, como se salienta na Figura 11a. Depois de todos os M valores de $\alpha'_k(s)$ terem sido calculados, devem ser divididos pela sua soma para obter-se $\alpha_k(s)$.

O procedimento repete-se até chegar-se ao fim da sequência recebida e, ter sido calculado $\alpha_N(0)$, lembrando que a treliça termina no estado nulo.

2.4.1.6 Cálculo de β

A quantidade β só pode ser calculada recursivamente depois de se ter recebido toda a sequência y. Conhecendo-se $\beta_k(s)$, o valor de $\beta_{k-1}(s')$ é calculado de maneira semelhante ao de $\alpha_k(s)$: vê-se que ramos saem do estado s', somam-se os correspondentes produtos de $\beta_k(s)$ e $\gamma_k(s)$ e divide-se pela soma $\sum_{s'} \beta_{k-1}(s')$. Por exemplo, na Fig. 10 vê-se que do estado $s' = 1$ saem dois ramos, um dirigido ao estado $s = 0$ e outro dirigido ao estado $s = 2$, como se realça na Fig. 11b. O processo repete-se até o cálculo de $\beta_0(0)$.

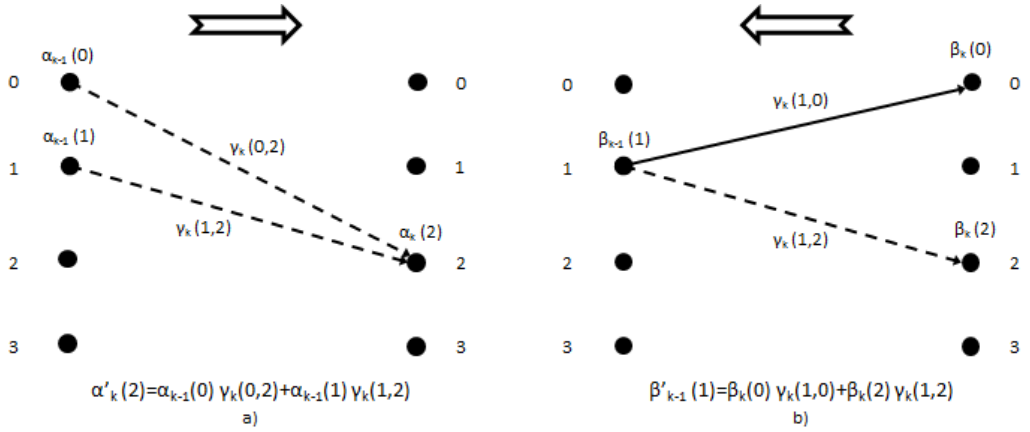

Figura 11 – Cálculo recursivo de α e β através da treliça.

2.4.1.7 Cálculo de $P(s', s, y)$ e $L(u_k|y)$

Com todos os valores de α, β e γ calculados, pode-se calcular a probabilidade conjunta $P(s', s, y) = \alpha_{k-1}(s)\ \gamma_k(s', s)\ \beta_k(s)/\sum_{Pk}$. Como exemplo, na Figura 12 é mostrado o cálculo de P(1,2,y) não normalizado, a partir de $\alpha_{k-1}(1)\ \gamma_k(1,2)\ \beta_k(2)$.

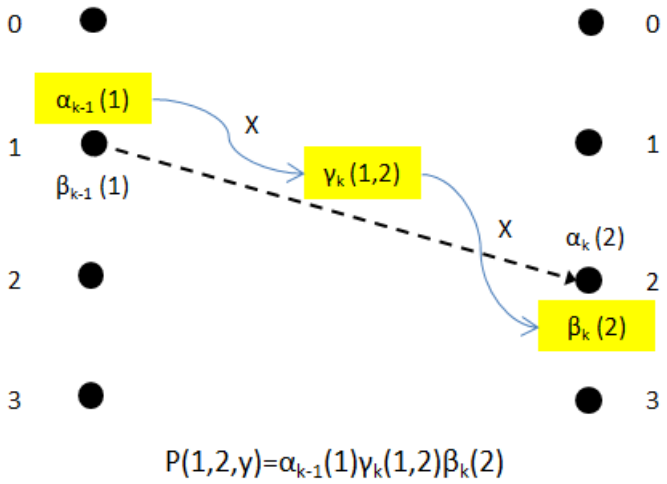

Figura 12 – A probabilidade *P(s',s, y)* como produto de três fatores *αγβ*.

Finalmente só resta calcular a *LLR a posteriori L(uk\y)*. Observando-se a treliça da Fig.10 pode ser visto que um bit de mensagem +1 origina as transições de estados seguintes: 0→2, 1→2, 2→3 e 3→3. São estas as transições R_1 da Eq. (10). As restantes quatro transições de estados, representados por traço contínuo (conjunto R_0), são devidas, é claro, a um bit de mensagem -1. Logo as quatro primeiras transições estão associadas ao numerador da Eq. (10) e as restantes ao denominador. Segue o equacionamento de *L(uk\y)* para os exemplos das figuras 11 e 12:

$$L(u_k|\mathbf{y}) = \ln \frac{\sum_{R_1} P(s',s,\mathbf{y})}{\sum_{R_0} P(s',s,\mathbf{y})} = \\ = \ln \frac{P(0,2,\mathbf{y}) + P(1,2,\mathbf{y}) + P(2,3,\mathbf{y}) + P(3,3,\mathbf{y})}{P(0,0,\mathbf{y}) + P(1,0,\mathbf{y}) + P(2,1,\mathbf{y}) + P(3,1,\mathbf{y})} \quad (27)$$

3. Caracterização do ruído impulsivo

É comum a simulação do ruído no canal de comunicação como sendo Gaussiano, ou seja, considerando que o canal é do tipo AWGN. Porém, para muitos canais físicos, o ruído encontrado e medido experimentalmente, tem comportamento não-Gaussiano, especialmente devido à natureza impulsiva da interferência eletromagnética produzida pelos dispositivos construídos pelo homem.

O modelo Gaussiano de representação do ruído foi extensivamente estudado por matemáticos e probabilistas e, o desenvolvimento de algoritmos baseados na suposição Gaussiana do ruído, é um procedimento bem conhecido. Contudo, esses algoritmos têm seu desempenho reduzido ao processarem sinais de comportamento não-Gaussiano. A disponibilidade de hardwares e softwares mais rápidos, nos dias atuais, dá as condições para o desenvolvimento de algoritmos mais eficientes para aplicações em tempo real sujeitas ao ruído impulsivo.

Dois modelos matemáticos de ruído impulsivo são bastante comentados na literatura: o modelo de Middleton e a distribuição alfa-estável. O fenômeno do ruído impulsivo foi primeiramente descrito por Middleton em [12, 13], onde ele deu um modelo para o ruído impulsivo em sistemas de comunicação. Ele dividiu os ruídos impulsivos em duas categorias: a) produzidos pelo homem, induzidos por outros equipamentos conectados à rede de comunicação; b) naturais, causados por fenômenos atmosféricos e estática solar, raios, manchas solares, etc.

No modelo de Middleton, o ruído é categorizado em três diferentes tipos (A, B e C). No modelo Middleton Classe A, o ruído de fundo (Gaussiano) mais o ruído impulsivo tem uma *pdf* igual a

$$p_\eta(v) = \sum_{k=0}^{\infty} \frac{e^{-A} A^k}{k!} \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{v^2}{2\sigma_k^2}\right), \tag{28}$$

sendo que

$$\sigma_k^2 = \left(1 + \frac{1}{\Gamma}\right)\left(\frac{(k/A) + \Gamma}{1 + \Gamma}\right)\sigma_b^2 \quad . \tag{29}$$

A expressão (28) pode ser considerada como uma soma ponderada de *pdfs* gaussianas onde o parâmetro A é chamado índice impulsivo, enquanto que o parâmetro Γ representa a relação ruído de fundo–ruído impulsivo. O parâmetro A representa a densidade de impulsos (de certa largura) em um período de observação.

No presente trabalho não foi feita a simulação computacional do ruído impulsivo pelo modelo de Middleton, mas apenas foi utilizado o modelo da distribuição alfa-estável simétrica ($S\alpha S$).

3.1 Distribuição de probabilidade alfa-estável

Mais recentemente foi proposto o modelo da distribuição alfa-estável simétrica ($S\alpha S$) [14] para o ruído impulsivo, modelo este recomendado para aplicações onde ruídos são tradicionalmente modelados como Gaussianos, porém onde picos repentinos ocorrem com maior frequência [2].

O TCL enuncia que a soma normalizada de um grande número de variáveis aleatórias independentes, igualmente distribuídas e de variância finita, converge para uma distribuição Gaussiana. Quando a restrição a que a variância seja finita é eliminada, ou seja, quando as variáveis aleatórias *i.i.d* podem ter variâncias tanto finitas quanto infinitas, o teorema pode ser generalizado como Teorema Central do Limite Generalizado e então, a soma converge para uma distribuição alfa-estável. Por isso, a distribuição Gaussiana é um caso particular da família de distribuições alfa-estável.

Com a exceção dos três casos especiais, que são a distribuição Gaussiana, a distribuição de Cauchy e a distribuição de Levy, não há expressões analíticas da função densidade de probabilidade (PDF) e da função de distribuição acumulada (CDF) para as distribuições alfa-estáveis e, por consequência, não existem todos os momentos. Em muitos problemas estatísticos, o primeiro momento E(X) e a variância $Var(X) = E(X^2) - E(X)^2$ são usados para descrever a distribuição. Contudo, eles geralmente não são úteis para as distribuições com cauda "pesada", porque as integrais nas expressões das suas esperanças podem divergir. No entanto, as PDFs e CDFs das distribuições alfa-estáveis podem ser obtidas por aproximações numéricas computacionais.

Embora não haja uma expressão fechada para a PDF, a função característica $\phi(t)$ sempre pode ser obtida. A forma da função característica $\phi(t)$, representada nas Eq. (30) e Eq. (31), foi obtida da referência [14]:

$$\phi(t) = Eexp(itX) = exp(-\gamma^{\alpha}|t|^{\alpha}[1 - i\beta sign(t)tan((\pi\alpha)/2)] + i\delta t), \text{ para } \alpha \neq 1, \quad (30)$$

$$\phi(t) = Eexp(itX) = exp(-\gamma|t|[1 + i\beta sign(t)(2/\pi)log|t|] + i\delta t), \text{ para } \alpha = 1, \quad (31)$$

onde

- α é o expoente característico satisfazendo o intervalo $0 < \alpha \leq 2$. O expoente característico controla o tempo de declínio das caudas da função densidade de probabilidade. Quanto mais "pesada" a cauda, mais impulsivo o ruído. Valores

baixos de α indicam ruídos mais impulsivos e valores altos indicam comportamento menos impulsivo. Quando $\alpha = 2$, a distribuição é gaussiana.

- δ é o parâmetro de localização ($-\infty < \delta < \infty$). Corresponde à média para $1 < \alpha \leq 2$ e à mediana para $0 < \alpha \leq 1$.
- γ é o fator de dispersão ($\gamma > 0$), que determina o espalhamento da densidade ao redor do parâmetro de localização. É similar à variância na densidade gaussiana e vale metade da variância quando $\alpha = 2$, que consiste no caso gaussiano.
- β é o índice de simetria ($-1 \leq \beta \leq 1$). Quando $\beta = 0$, a distribuição é simétrica ao redor do parâmetro de localização.

O caso $\alpha = 2$ e $\beta = 0$ corresponde à distribuição Gaussiana, enquanto que $\alpha = 1$ e $\beta = 0$ corresponde à distribuição de Cauchy. As funções densidade de probabilidade destas duas distribuições são dadas, respectivamente, por

$$f_{\alpha=2}(\gamma, \lambda, x) = \frac{1}{\sqrt{4\pi\gamma}}\, exp\left\{-\frac{(x-\lambda)^2}{4\gamma}\right\} \tag{32}$$

e

$$f_{\alpha=1}(\gamma, \lambda, x) = \frac{\gamma}{\pi[\gamma^2 + (x-\lambda)^2]}. \tag{33}$$

Em [15] é proposto um algoritmo para o cálculo numérico da função distribuição de probabilidade. Utilizando o código Matlab® proposto na referência, foram obtidos os gráficos das *pdfs* para diferentes valores do expoente característico α e considerando β = 0, o que produz distribuições simétricas. A Figura 13 mostra os gráficos obtidos.

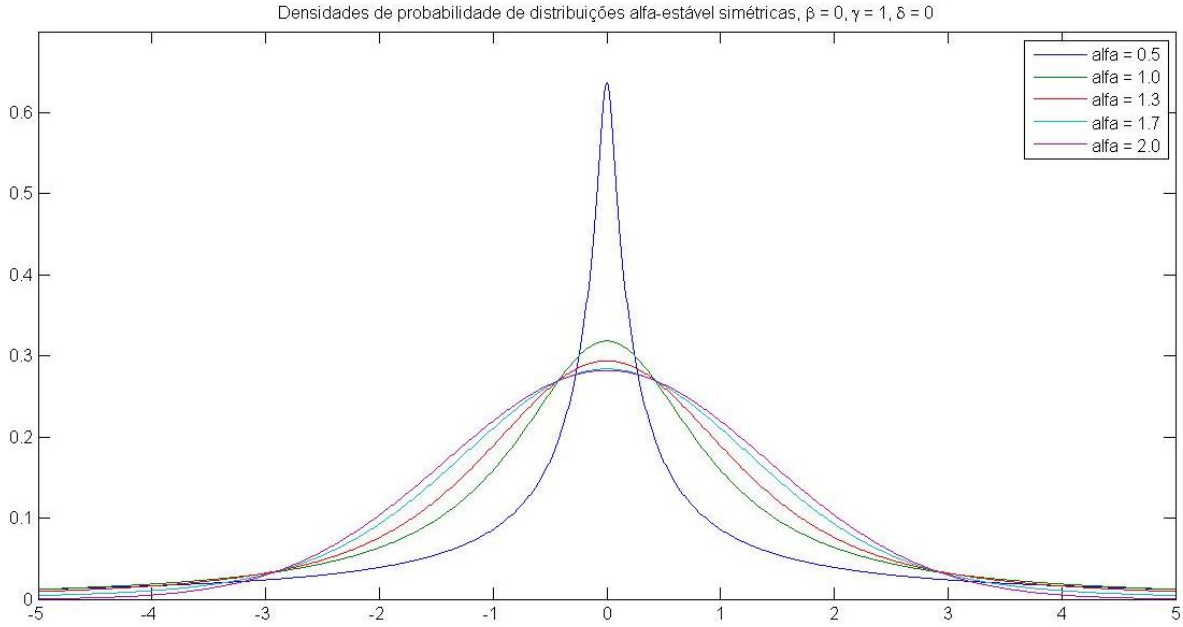

Figura 13 – Gráfico das funções densidade de probabilidade da distribuição α-estável simétricas para diferentes valores de alfa.

As variáveis aleatórias de um processo estocástico com distribuição alfa-estável podem ser simuladas pela equação abaixo [16].

$$\varphi(t) = \frac{\sin(\alpha\theta)}{\cos(\theta)^{1/\alpha}}\left[\frac{\cos((\alpha-1)\theta)}{W}\right]^{(1-\alpha)/\alpha} \tag{34}$$

onde $\theta = \pi(u_1(t) - 0{,}5)$, $W = -\log(u_2(t))$, $u_1(t)$ e $u_2(t)$ são variáveis aleatórias uniformemente distribuídas no intervalo [0,1], com média 0,5 e α é o coeficiente característico definido no intervalo $1 < \alpha < 2$.

Para a distribuição Cauchy, com $\alpha = 1$, a função amostra de um processo estocástico com distribuição alfa-estável é calculada pela equação abaixo [16].

$$\varphi(t) = \tan(\theta) \tag{35}$$

Neste trabalho utiliza-se um código em Matlab® para gerar ruído com distribuição alfa-estável. O código implementa as equações (34) e (35). Através do programa em

Matlab® foram gerados gráficos de amostras de ruído com distribuição *SαS* para diferentes valores de α, conforme é mostrado na Figura 14.

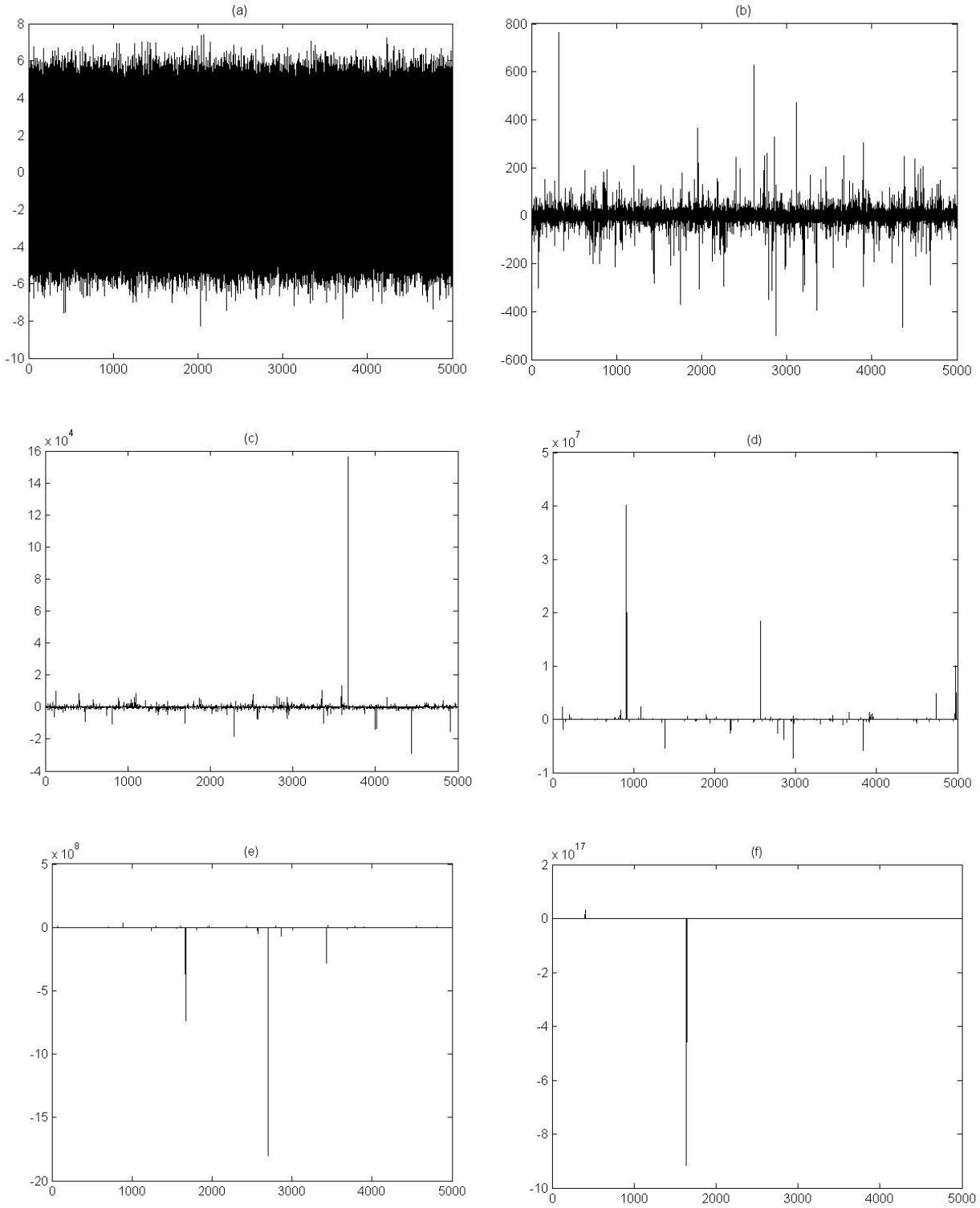

Figura 14 – Gráficos de amostras de ruído com distribuição alfa-estável para a) α=2 (Gaussiana); b) α=1,95 c) α=1,5; d) α=1 (Cauchy); e) α=0,85 e f) α=0,45

3.2 Medida da potência do ruído para a distribuição alfa-estável

Devido ao segundo momento de uma sequência aleatória com distribuição alfa-estável não convergir quando o parâmetro α é menor que 2, a variância não pode ser usada como uma medida da potência do ruído impulsivo, da mesma maneira que ela é usada para determinar a potência do ruído Gaussiano. Para simular o desempenho do código turbo é necessária uma medida da relação sinal-ruído e, consequentemente, da potência do ruído.

Nas seções seguintes serão examinadas a relação energia de bit por densidade espectral de potência do ruído (Eb/N0) para o canal AWGN e, a sugestão para a medição da relação sinal-ruído para o canal sujeito ao ruído impulsivo com distribuição alfa-estável simétrica, denominada Eb/N0 generalizada *(GEbN0 – Generalized Eb/N0*), proposta em [17].

3.2.1 Eb/N0 para o canal AWGN

Supondo um sinal modulado x, transmitido por um canal AWGN. O sinal recebido é y = ax + n, onde "a" é um fator de escala de amplitude do canal, que é usualmente igual a "1" para o canal AWGN e "n" é o ruído representado por uma variável aleatória gaussiana de média zero e variância σ^2.

Para o canal AWGN a variância do ruído é dada por $\sigma^2 = N0/2$, onde N0 é a densidade espectral de potência do ruído.

Para esquemas de modulação M-ários como o M-PSK, incluindo o BPSK, a energia de símbolo é dada por Es = RmRcEb, onde Es é a energia de símbolo do sinal modulado, Rm = log2(M), Rc é a taxa do código e Eb é a energia por informação de bit. Para a modulação BPSK, M é igual a 2, para QPSK, M = 4, para 16-QAM, M =16, etc.

Assumindo Es normalizada igual a 1, Eb/N0 pode ser representada pela equação abaixo:

$$\frac{Eb}{N0} = \frac{Es}{Rm.Rc.N0} = \frac{\mathrm{Es}}{\mathrm{Rm.Rc.}2\sigma^2} = \frac{1}{2.\mathrm{Rm.Rc.}\sigma^2}\,. \qquad (36)$$

Isolando-se a variância σ^2 na Eq. 36 e posteriormente aplicando a raiz quadrada nos dois lados da equação, encontramos o desvio padrão (σ) para o canal AWGN.

$$\sigma = \frac{1}{\sqrt{2.Rm.Rc.\left(\frac{Eb}{N0}\right)}} \quad (37)$$

3.2.2 Eb/N0 generalizada

O parâmetro dispersão (γ) da distribuição alfa-estável tem sido usado como uma medida proporcional da potência do ruído. Em [17] sugere-se uma equação para a medida da relação Eb/N0, denominada Eb/N0 generalizada (GEbN0), relacionando os parâmetros γ e α da distribuição alfa-estável, conforme abaixo:

$$GEbN0 = \frac{Eb}{4\gamma^{2/\alpha}} \quad . \quad (38)$$

Considerando $E_s = R_m R_c E_b$ e assumindo Es normalizada igual a "1" e isolando-se γ na Eq. (38), encontra-se o valor da dispersão,

$$\gamma = \frac{1}{\sqrt{(4.Rm.Rc.GEbN0)^{\alpha}}} \quad . \quad (39)$$

4. Simulação

Neste capítulo são apresentados os resultados das simulações computacionais realizadas. Para a simulação foi utilizado o código Matlab®, elaborado parcialmente pelo professor e pesquisador Mathew Valenti da Universidade Virginia Tech, que implementa um código turbo para um canal AWGN e sistema de modulação BPSK.

No código Matlab® podem ser configurados os seguintes parâmetros de simulação: tamanho do quadro (nº de bits do quadro); polinômio gerador do codificador; escolha entre duas taxas de código, ou seja, taxa igual a 1/2 quando utilizado o puncionamento de bits de paridade ou, taxa igual a 1/3, sem utilizar o puncionamento; critério de parada através do número limite de quadros com erros; o range de Eb/N0; algoritmo de decodificação (SOVA ou MAP) e o número de iterações da decodificação. Ao código foram adicionados os seguintes complementos: um gerador de amostras de ruído impulsivo com distribuição alfa-estável simétrica, podendo-se variar o coeficiente α; a adequação da medida da relação

sinal-ruído ao ruído impulsivo, utilizando a Eq. 39; critério de parada baseado no número de bits enviados e um laço para controlar o número de realizações da simulação.

4.1 Simulação para o canal AWGN

Nesta simulação avaliou-se o desempenho do código turbo em um canal AWGN puro, ou seja, sem a adição do ruído impulsivo ao canal. O desempenho do código no canal AWGN serviu para obter critérios de comparação com os resultados obtidos nas demais simulações. As simulações seguintes adicionam o ruído impulsivo ao canal. São efetuadas simulações para diferentes valores do expoente característico α da distribuição SαS, iniciando com α igual a 2 e, posteriormente, acentuando-se o caráter impulsivo do ruído, por meio da diminuição do valor de α.

A simulação para o canal AWGN foi realizada com os parâmetros destacados no quadro abaixo, ou seja, Eb/N0 variando de 0 a 4 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 8 iterações e um número de realizações igual a 10.

A Figura 15 mostra o gráfico Eb/N0(dB) x BER gerado.

```
-----------------------------------------------------------------------------------------------
=== Log-MAP decoder ===
Frame size =   1024
code generator:
   1     1     1
   1     0     1
Punctured, code rate = 1/2
iteration number =       8
terminate frame errors =    100
Eb / N0 (dB)= 0.00 0.50 1.00 1.50 2.00 2.50 3.00 3.50 4.00
N° Simulações = 10
-----------------------------------------------------------------------------------------------
```

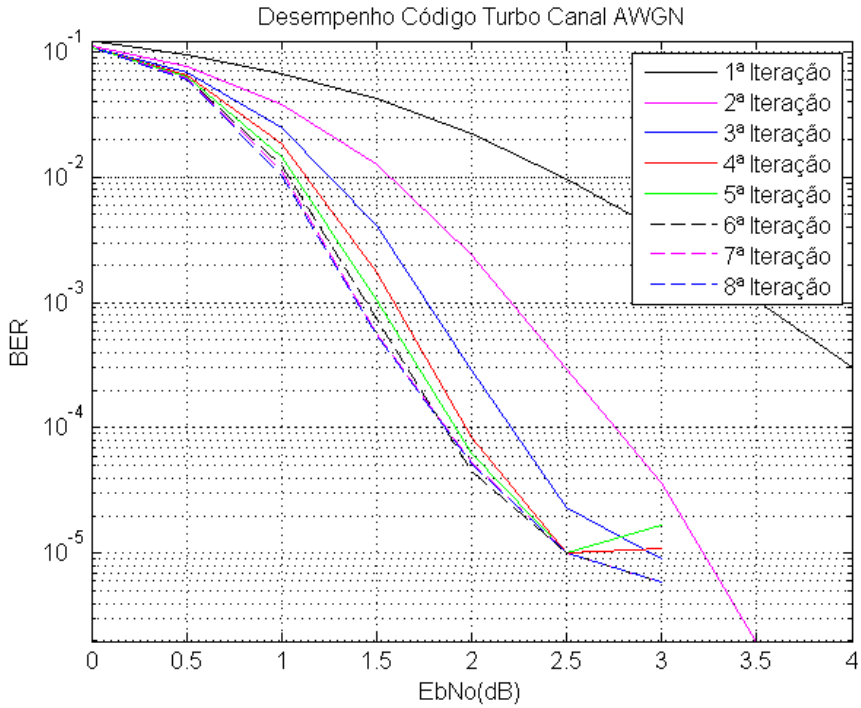

Figura 15 – Gráfico EbN0(dB) x BER para o canal AWGN

Resultados

Nesta simulação foi verificado o desempenho do código turbo em um canal AWGN. A Figura 15 mostra as curvas BER x Eb/N0 para as diferentes iterações do decodificador. Pode-se observar que uma BER da ordem de 10^{-5} é alcançada com Eb/N0 < 3 dB, a partir da 4ª iteração. Um maior ganho de codificação é observado nas primeiras três iterações. Observa-se que a BER melhora a cada iteração, porém o ganho vai diminuindo entre uma iteração e outra, o que não justifica o esforço do processamento computacional envolvido em um número muito elevado de iterações. O gráfico apresenta distorções para as curvas a partir da 4ª iteração, que podem ser causadas provavelmente por instabilidade numérica durante os cálculos realizados pelo algoritmo BCJR-MAP.

4.2 Simulação com distribuição alfa-estável (α = 2)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável simétrica e com expoente característico α igual a 2. Para esse valor do expoente característico, conforme exposto anteriormente, a distribuição alfa-estável equivale à distribuição normal e, portanto, é esperado um resultado similar ao obtido na seção 4.1. O código Matlab® foi alterado para que a medida da potência do ruído fosse ajustada ao ruído impulsivo, conforme exposto na seção 3.2.2, ou seja, foi utilizado o cálculo da variância do ruído definido na Eq. 39.

A simulação foi realizada com os mesmos parâmetros da seção anterior, ou seja, Eb/N0 variando de 0 a 4 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 8 iterações e um número de realizações igual a 10. A Figura 16 mostra os gráficos EbN0(dB) x BER gerados.

```
---------------------------------------------------------------------------------------
 === Log-MAP decoder
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       8
 terminate frame errors =    100
 Eb / N0 (dB) = 0.00 0.50 1.00 1.50 2.00 2.50 3.00 3.50 4.00
 Alfa = 2.0
 N° Simulações = 10
---------------------------------------------------------------------------------------
```

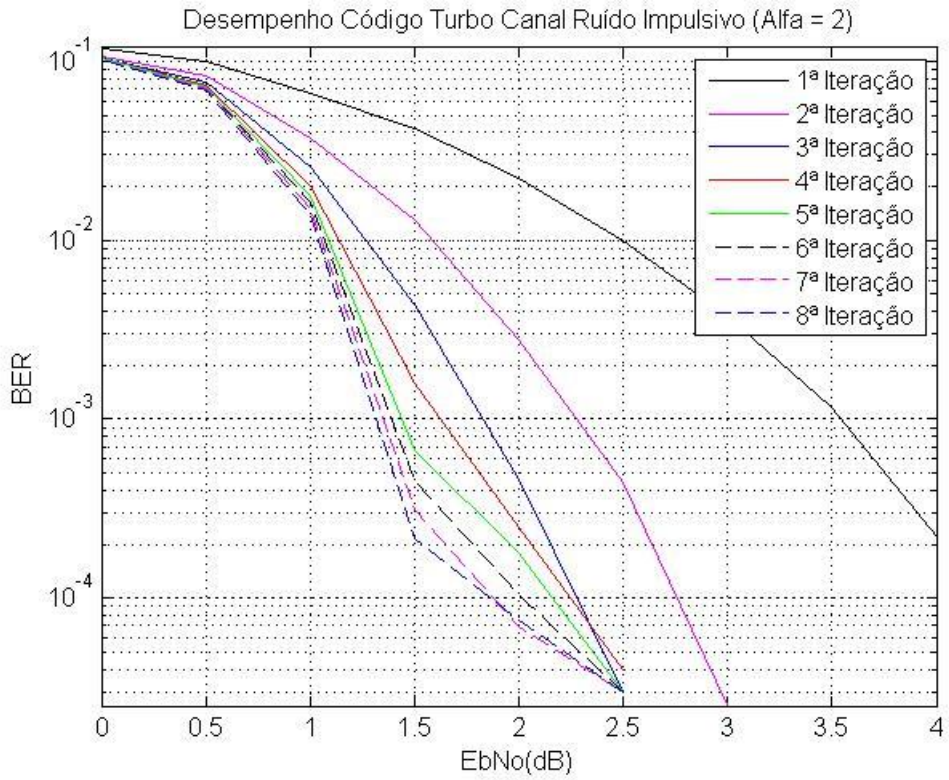

Figura 16 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=2)

Resultados

A Figura 16 mostra as curvas BER x Eb/N0 obtidas, na qual se observa o ganho do código ao passar das iterações. O resultado mostra um comportamento similar ao observado para o canal AWGN, conforme esperado. Observa-se um maior ganho de codificação nas primeiras três iterações e uma BER similar à obtida para o canal AWGN.

O resultado demonstra que a utilização da Eb/N0 Generalizada foi adequada para a simulação do comportamento Gaussiano do ruído, ou seja, quando o expoente característico da distribuição *SαS* é igual a 2.

4.3 Simulação com distribuição alfa-estável (α = 1,98)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,98 e parâmetros destacados no quadro abaixo.

A simulação foi realizada com Eb/N0 variando de 0 a 4 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 8 iterações e um número de realizações igual a 10. A Figura 17 mostra os gráficos EbN0(dB) x BER gerados.

```
-----------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       8
 terminate frame errors =    100
 Eb / N0 (dB) =  0.00 0.50 1.00 1.50 2.00 2.50 3.00 3.50 4.00
 Alfa = 1,98
 N° Simulações = 10
-----------------------------------------------------------------
```

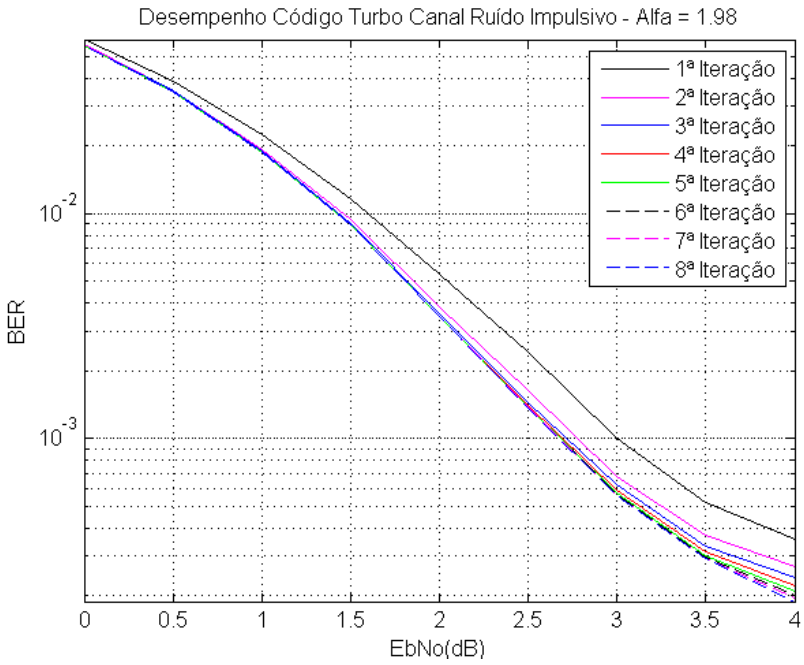

Figura 17 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,98).

Resultados

Ao diminuir ligeiramente α, introduziu-se ruídos impulsivos caracterizados por picos de alta amplitude. Isto se reflete na degradação do desempenho do código em relação à simulação anterior com α = 2. Porém, entre a primeira e a segunda iteração, o código continua apresentando um ganho expressivo. A partir da terceira iteração, praticamente não se observa aumento no ganho.

4.4 Simulação com distribuição alfa-estável (α = 1,90)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,90 e os parâmetros destacados no quadro abaixo. Foram feitas 10 realizações.

A simulação foi realizada com Eb/N0 variando de 0 a 4 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 8 iterações e um número de realizações igual a 10. A Figura 18 mostra os gráficos EbN0(dB) x BER gerados.

```
-----------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       8
 terminate frame errors =    100
 Eb / N0 (dB) = 0.00 0.50 1.00 1.50 2.00 2.50 3.00 3.50 4.00
 Alfa = 1,90
 N° Simulações = 10
```

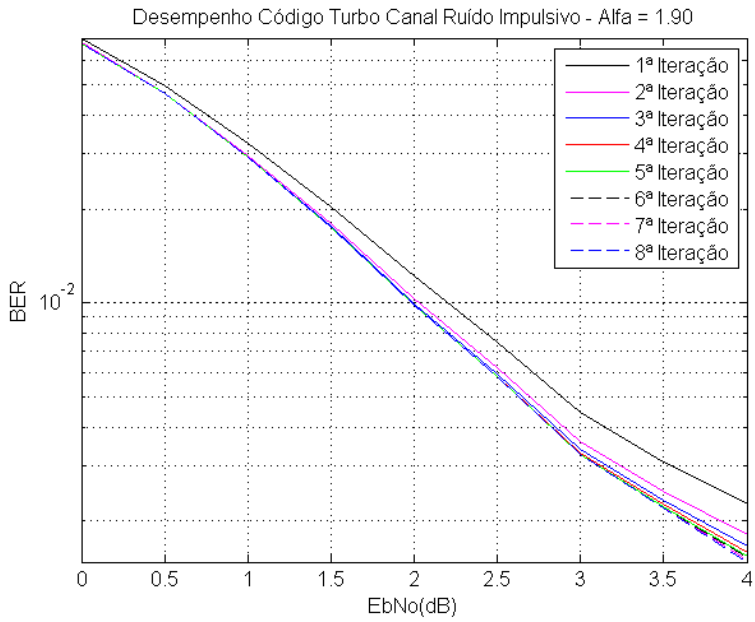

Figura 18 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,90).

Resultados

Aumentando o caráter impulsivo do ruído fica evidente a degradação na BER, que não atinge 10^{-3} para uma Eb/N0 de 3dB. Pode-se verificar o desempenho do código com o passar das iterações, porém o ganho maior acontece entre a primeira e a segunda iteração.

4.5 Simulação com distribuição alfa-estável (α = 1,80)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,80 e os parâmetros destacados no quadro abaixo.

A simulação foi realizada com Eb/N0 variando de 0 a 8 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 5 iterações e um número de realizações igual a 30. A Figura 19 mostra os gráficos EbN0(dB) x BER gerados.

```
------------------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       5
 terminate frame errors =    100
 Eb / N0 (dB) =     0.00      0.50      1.00      1.50      2.00
2.50      3.00      3.50      4.00      4.50      5.50      5.50
6.00      6.50      7.00      7.50      8.00
 Alfa = 1.80
 N° Simulações = 30
```

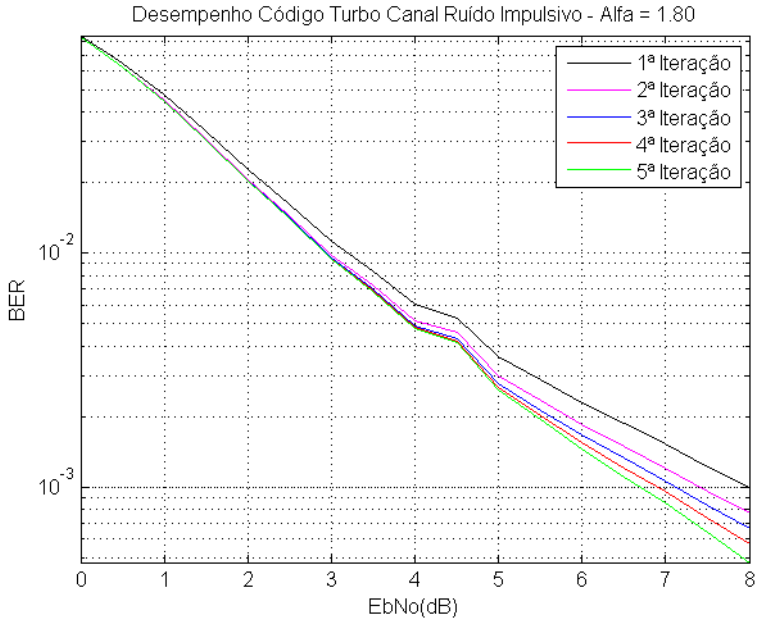

Figura 19 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,80).

Resultados

A simulação foi realizada para uma Eb/N0 de até 8 dB, para verificar o comportamento do código com relação sinal-ruído mais alta, diante do fato que, para α igual a 1,80, a amplitude do ruído impulsivo será mais elevada.

Conforme esperado, no caso de α = 1, 80, a grande de quantidade de ruídos impulsivos de elevada amplitude degrada o sinal e, uma BER de 10^{-3} só é atingida com uma Eb/N0 de 7 dB e quatro iterações.

4.6 Simulação com distribuição alfa-estável (α = 1,70)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,70 e parâmetros destacados no quadro abaixo.

A simulação foi realizada com Eb/N0 variando de 0 a 10 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 5 iterações e um número de realizações igual a 50. A Figura 20 mostra os gráficos EbN0(dB) x BER gerados.

```
-------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       5
 terminate frame errors =    100
 Eb / N0 (dB) =     0.00      0.50      1.00      1.50      2.00
2.50      3.00      3.50      4.00      4.50      5.00      5.50
6.00      6.50      7.00      7.50      8.00      8.50      9.00
9.50     10.00
Alfa = 1.7
N° Simulações = 50
```

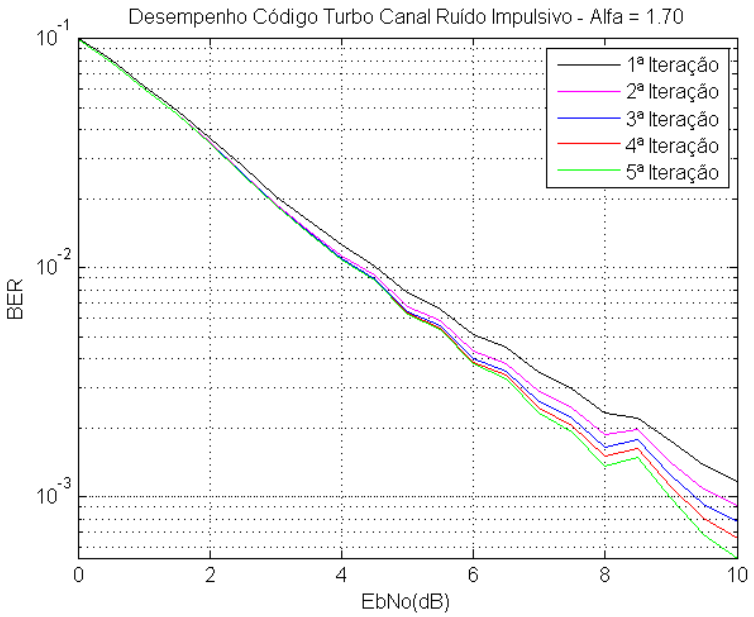

Figura 20 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,70).

Resultados

Uma BER de 10^{-3} só é atingida com uma Eb/N0 de aproximadamente 9 dB e cinco iterações. Ou seja, o ganho do código diminui em 2 dB alterando o valor de alfa de 1,80 para 1,70.

4.7 Simulação com distribuição alfa-estável (α = 1,50)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,50 e parâmetros destacados no quadro abaixo.

A simulação foi realizada com Eb/N0 variando de 0 a 10 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 5 iterações e um número de realizações igual a 50. A Figura 21 mostra os gráficos EbN0(dB) x BER gerados.

```
-----------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       5
 terminate frame errors =    100
 Eb / N0 (dB) =     0.00      0.50      1.00      1.50      2.00
2.50      3.00      3.50      4.00      4.50      5.00      5.50
6.00      6.50      7.00      7.50      8.00      8.50      9.00
9.50     10.00
Alfa = 1.5
N° Simulações = 50
-----------------------------------------------------------------
```

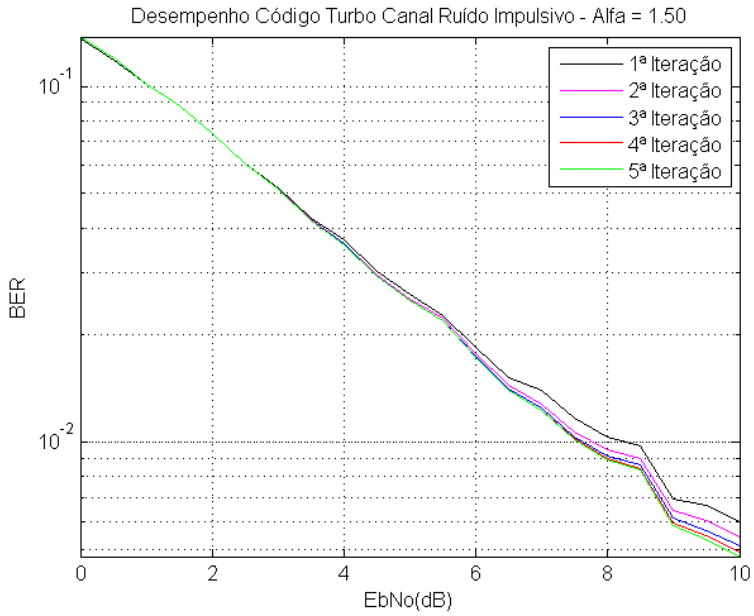

Figura 21 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,50).

Resultados

O sinal fica bastante degradado e o código só começa a ter algum desempenho com uma Eb/N0 acima de 6 dB. Em 10 dB, mesmo com 5 iterações a BER fica abaixo de 10^{-3}.

4.8 Simulação com distribuição alfa-estável (α = 1,30)

Nesta simulação, o código foi configurado para utilizar as amostras do ruído com distribuição alfa-estável, com α = 1,30 e parâmetros destacados no quadro abaixo. Foram feitas 50 realizações.

A simulação foi realizada com Eb/N0 variando de 0 a 10 dB, quadro de 1024 bits, taxa de código igual a 1/2, sequências geradoras $g_1 = 7$ e $g_2 = 5$, critério de parada de 100 quadros com erros, decodificação MAP com 5 iterações e um número de realizações igual a 50. A Figura 22 mostra os gráficos EbN0(dB) x BER gerados.

```
------------------------------------------------------------------
 === Log-MAP decoder ===
 Frame size =   1024
 code generator:
     1     1     1
     1     0     1
 Punctured, code rate = 1/2
 iteration number =       5
 terminate frame errors =    100
 Eb / N0 (dB) =     0.00      0.50      1.00      1.50      2.00
2.50      3.00      3.50      4.00      4.50      5.00      5.50
6.00      6.50      7.00      7.50      8.00      8.50      9.00
9.50     10.00
Alfa = 1.3
N° Simulações = 50
------------------------------------------------------------------
```

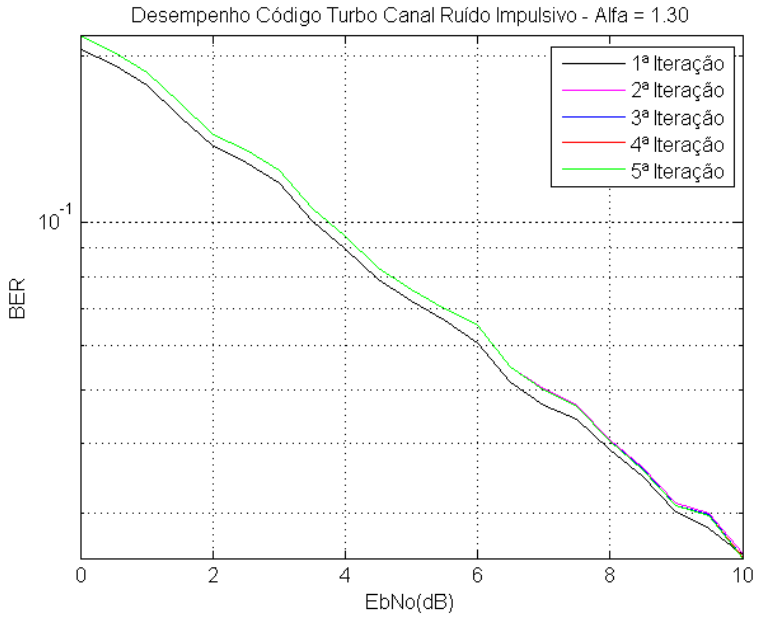

Figura 22 – Gráficos EbN0(dB) x BER para ruído com distribuição alfa-estável (α=1,30).

Resultados

Nesta situação, com ruídos impulsivos de altíssimas amplitudes, o código praticamente não funciona e, mesmo com uma Eb/N0 de 10dB, não se alcança uma BER de 10^{-2}, o que indica pelo menos um erro a cada 100 bits.

5. Conclusão

Códigos turbo são uma classe de códigos que exibem propriedades de códigos de bloco de grande comprimento, utilizando codificadores recursivos. A performance do código é dependente, dentre outros requisitos, do projeto do seu entrelaçador, que deve garantir o peso de Hamming adequado em pelo menos um dos códigos constituintes. Decodificadores suaves são usados para permitir que a probabilidade *a posteriori* seja passada, entre as iterações, de um ao outro elemento decodificador.

Um código turbo de taxa 1/2 em um canal AWGN e, usando-se modulação BPSK, oferece um ganho de codificação de 7 dB em relação ao canal não codificado, a uma BER de 10^{-5}.

O desempenho do código turbo em um canal ruidoso foi analisado, através de simulação computacional. O canal proposto no trabalho tem características impulsivas, modeladas através da distribuição de probabilidade alfa-estável simétrica.

Foi verificada a diminuição do desempenho do código à medida que foi aumentada a impulsividade do ruído no canal. Percebeu-se a necessidade de adaptação no código para melhor desempenho com o ruído impulsivo, pois o código turbo clássico é projetado para funcionar com o canal AWGN. Na referência [19] é proposta uma modificação no algoritmo BCJR-MAP para adequá-lo ao ruído impulsivo, sendo que este é modelado através da distribuição alfa-estável simétrica.

Outra proposta para melhorar o desempenho do código é a concatenação em série do código turbo com um código Reed Solomon, que apresenta bom desempenho em sistemas sujeitos à ocorrência de erros em rajada.

O trabalho permitiu o aprofundamento do conhecimento sobre códigos corretores de erro avançados, tanto do ponto de vista teórico, quanto da sua simulação por meio de ferramenta computacional. Conceitos centrais do curso de Engenharia da Informação, especialmente nas áreas de teoria da informação e códigos, sinais aleatórios e comunicações digitais, puderam ser mais bem apreendidos e investigados.

6. Referências bibliográficas

[1] KAY, S., "Intuitive Probability and Random Processes Using MATLAB", Springer, 2006.

[2] WANG, J.: ZHOU, T. - Alpha-Stable Channel Capacity - IEEE Communications Letters, v. 15, nº. 10, Outubro/2011

[3] PANAYIOTIS, G. Georgiou; PANAGIOTIS, Tsakalides. "Alpha-Stable Modeling of Noise and Robust Time-Delay Estimation in the Presence of Impulsive Noise", IEEE Transactions on Multimedia, vol. 1, no 3, pp. 291-301, Setembro 1999.

[4] SHANNON, C. E. A mathematical theory of communication. The Bell System Technical Journal. Jul/2012.

[5] SKLAR, Bernard ; HARRIS, Fredric J. "The ABCs of Linear Block Codes – An intuitive treatment of error detection and correction" , IEEE Signal Processing Magazine, Julho, 2004.

[6] BERROU, C; GLAVIEUX, A; THITIMAJSHIMA, O. " Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes". ICC'93.

[7] LIN, S.; COSTELLO, D. Error Control Coding (2nd Edition). Pearson Prentice, 2004

[8] VITERBI, A. J. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". IEEE Transactions on Information Theory 13(2): 260–269. Abr/1967

[9] JOHNSON, Sarah J. "Turbo, Low-Density Parity-Check and Repeat –Accumulate Codes ", Cambridge Press, 2009.

[10] BENEDETTO S., DIVSALAR D., MONTOROSI G., POLLARA F., "Serial concatenation of interleaved codes: Performance analysis, design and iterative decoding". IEEE Trans. Inform. Theory, vol. 44, pp. 909–926, Maio 1998.

[11] L. R. Bahl, J. Cocke, F. Jelinek, e J. Raviv. " Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate, IEEE Transactions on Information Theory, vol. 20, nº. 2, pags. 284–287, março 1974.

[12] MIDDLETON, D, An introduction to statistical communication theory. McGraw-Hill New York, 1960, vol. 960.

[13] MIDDLETON, D. Statistical-physical models of electromagnetic interference. IEEE Transactions on electromagnetic compatibility, vol. 19, n° 3, pp. 106-127, 1977.

[14] TSIHRINTZIS G. A.; NIKIAS, C. L. "Fast estimation of the parameters of alpha-stable impulsive interference", IEEE Transactions on Signal Processing, vol. 44, no 6, pp. 1492-1503, Junho 1996.

[15] http://math.bu.edu/people/mveillet/html/alphastablepub.html#17 – Acessado em 10/08/2015.

[16] NOLAN, John P. Stable Distributions – Models for Heavy Tailed Data. American University, 2014, pag. 21.

[17] LAGUNA-SANCHEZ, Gerardo; LOPEZ-GUERRERO, Miguel. "On the use of alpha-stable distributions in noise modeling for PLC", IEEE Transactions on Power Delivery, vol. XX, no X, Dezembro 201X.

[18] WU, Yufei. Virginia Tech, MPRG Lab, Novembro, 1998.

[19] SHAFIEIPOUR, M.; LIM H.; CHUAH T. "Decoding of Turbo Codes in Symmetric Alpha-Stable Noise", International Scholarly Research Network ISRN Signal Processing, 2011.