UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM SISTEMA DE MÚLTIPLO ACESSO BASEADO NA TRANSFORMADA NUMÉRICA DE FOURIER

por

MARIA APARECIDA DA SILVA

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR: RICARDO MENEZES CAMPELLO DE SOUZA, Ph.D.

Recife, Julho de 2011.

Catalogação na fonte
Bibliotecária Margareth Malta, CRB-4 / 1198

S586u Silva, Maria Aparecida da.
Um sistema de múltiplo acesso baseado na transformada numérica de Fourier / Maria Aparecida da Silva. - Recife: O Autor, 2013.
ix, 89 folhas, il., gráfs., tabs.

Orientador: Prof. Dr. Ricardo Menezes Campello de Souza.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Elétrica, 2013.
Inclui Referências.

1. Engenharia Elétrica. 2. Transformadas numéricas. 3. Comunicação multiusuário. 4. Códigos corretores de erro. I. Souza, Ricardo Menezes Campello de. (Orientador). II. Título.

UFPE

621.3 CDD (22. ed.) BCTG/2013-252

Universidade Federal de Pernambuco

Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO DO MESTRADO ACADÊMICO DE

MARIA APARECIDA DA SILVA

TÍTULO

"UM SISTEMA DE MÚLTIPLO ACESSO BASEADO NA TRANSFORMADA NUMÉRICA DE FOURIER"

A comissão examinadora composta pelos professores: RICARDO MENEZES CAMPELLO DE SOUZA, DES/UFPE, HÉLIO MAGALHÃES DE OLIVEIRA, DES/UFPE e JULIANO BANDEIRA LIMA, POLI/UPE sob a presidência do primeiro, consideram a candidata **MARIA APARECIDA DA SILVA APROVADA.**

Recife, 29 de julho de 2011.

RAFAEL DUEIRE LINS Coordenador do PPGEE	**RICARDO MENEZES CAMPELLO DE SOUZA** Orientador e Membro Titular Interno
JULIANO BANDEIRA LIMA Membro Titular Externo	**HÉLIO MAGALHÃES DE OLIVEIRA** Membro Titular Externo

AGRADECIMENTOS

Primeiramente, agradeço a Deus, Força Suprema do Universo. Ao meu orientador, Professor Ricardo Menezes Campello de Souza, pelo apoio e orientação na elaboração desta dissertação, mesmo nos momentos mais difíceis e cuja competência e dedicação sempre me serviram de exemplo.

Agradeço-o principalmente por ter aceitado ser meu orientador de mestrado.

Aos demais professores do Departamento de Eletrônica e Sistemas com os quais tive o prazer de estudar: Hélio Magalhães de Oliveira, Valdemar Cardoso da Rocha Jr. e Cecílio José Lins Pimentel, que muito contribuíram para minha formação, e a Andréa Tenório pela ajuda constante.

Aos professores Juliano Bandeira Lima, pelas contribuições imprescindíveis e ensinamentos recebidos desde a minha graduação em Engenharia de Telecomunicações, e Raimundo Corrêa de Oliveira, que aceitou ajudar-me e ouvia pacientemente minhas dúvidas, colaborando sempre com críticas construtivas na utilização da linguagem computacional *Matlab*®.

Meu reconhecimento especial a Leandro, por sua ajuda, sem a qual este trabalho não teria sido possível.

Também não posso esquecer os meus amigos da CLARO, pelo coleguismo e apoio nos momentos de ausência do trabalho, especialmente ao coordenador Adonis Pedro da Silva Neto por sua compreensão nos momentos mais críticos.

MARIA APARECIDA DA SILVA

Universidade Federal de Pernambuco
Julho de 2011

Resumo da Dissertação apresentada a UFPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

UM SISTEMA DE MÚLTIPLO ACESSO BASEADO NA TRANSFORMADA NUMÉRICA DE FOURIER

Maria Aparecida da Silva

Julho/2011

Orientador: Prof. Ricardo Menezes Campello de Souza, Ph.D.

Área de Concentração: Comunicações

Palavras-chaves: Transformadas numéricas, comunicação multiusuário, códigos corretores de erro.

Numero de Páginas: 97

Esta dissertação apresenta um estudo de um sistema de comunicação multiusuário proposto recentemente na literatura, o qual se baseia na autoestrutura da transformada numérica de Fourier e usa autossequências desta transformada como assinaturas de usuários. O trabalho descreve a implementação do sistema por meio de simulação computacional, bem como apresenta uma avaliação de desempenho do mesmo. A simulação, implementada por meio do aplicativo Simulink/MatLab™, considera o caso em que dois usuários transmitem informação simultaneamente por meio do canal GF(p) aditivo, em presença de ruído gaussiano branco aditivo. Em uma primeira análise, por meio de uma avaliação do desempenho apresentado pelo sistema, em termos de taxa de erro de símbolo *versus* relação sinal-ruído, é mostrado que o mesmo é capaz de separar as informações dos usuários, mesmo sem o uso de codificação de canal. Para avaliar o desempenho do sistema com codificação de canal é usada uma família de códigos corretores de erro recentemente construída, a dos códigos de Fourier, cujas palavras são autossequências da transformada numérica de Fourier. O desempenho do sistema codificado é avaliado e comparado com o desempenho apresentado pelo mesmo sem o uso de codificação de canal.

Abstract of Dissertation presented to UFPE as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

A MULTIPLE ACCESS SYSTEM BASED ON THE FOURIER NUMBER THEORETIC TRANSFORM

Maria Aparecida da Silva

Julho/2011

Supervisor: Prof. Ricardo Menezes Campello de Souza, Ph.D.

Area of Concentration: Communications

Keywords: Number Theoretic Transforms, Multiuser Communication, Error Correcting Codes.

Number of pages: 97

This dissertation presents a study of a multiuser communication system proposed recently in the literature, which is based on the eigenstructure of the Fourier number theoretic transform and uses eigensequences of this transform as user signatures. The work describes the implementation of the system, through computer simulation, and presents an evaluation of its performance. The simulation is implemented via Simulink/MatLab™, and it considers the case in which two users transmit information simultaneously through the GF(p) additive channel, in the presence of additive white Gaussian noise.

Initially, by evaluating the performance presented by the system, in terms of the symbol error rate versus signal-to-noise ratio, it is shown that the system is capable of separating the user information, even without the use of channel coding. In order to evaluate the performance of the system with channel coding, a family of error correcting codes recently constructed, namely the Fourier codes, is used, whose codewords are eigensequences of the Fourier number theoretic transform. The performance of the coded system is evaluated and compared to the performance presented by the system without the use of error control.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Em Engenharia, as transformadas definidas sobre corpos finitos têm sido empregadas em diversas aplicações. Áreas como processamento de sinais e códigos corretores de erros são beneficiadas pelo uso destas ferramentas, que proporcionam vantagens relacionadas à precisão e complexidade [1], [2], [3].

A transformada de corpo finito mais conhecida é a de Fourier (FFFT, do inglês *finite field Fourier transfom*), que foi introduzida por Pollard em 1971 em sua versão numérica [4]. As chamadas transformadas numéricas realizam uma espécie de mapeamento de um vetor com componentes em $GF(p)$ num vetor transformado cujas componentes também estão em $GF(p)$, o campo de *Galois* de ordem p. Esta transformação possui propriedades análogas às da transformada discreta de Fourier (DFT, do inglês *discrete Fourier transform*), tais como linearidade, deslocamento no tempo e convolução cíclica [5].

Além da FFFT, outras transformadas de corpos finitos têm sido propostas. Um exemplo disso é a transformada de Hartley de corpo finito (THCF), para a qual se propôs aplicações no projeto de sistema de multiplexação digital, em sistemas de múltiplo acesso e no espalhamento espectral de sequências [5],[6],[7],[8],[9],[10]. Uma outra importante classe de transformadas, é a das transformadas trigonométricas discretas (DTTs, do inglês *discrete trigonometric transforms*). As funcionalidades destas transformadas continuam sendo estudadas, fato refletido na diversidade de artigos em que são propostos sistemas e técnicas baseadas nas mesmas [11], [12], [13], [14]. As duas primeiras transformadas trigonométricas sobre corpos finitos (FFTT, do inglês *finite field trigonometric transform*), a do cosseno e a do seno tipo I, foram introduzidas por Campello de Souza et al. [15], [16]. Posteriormente, as demais das transformadas, completando um total de 16 transformadas trigonométricas, foram construídas por J. B. Lima [17]. As FFTTs têm diversas aplicações, por exemplo, marca d'água digital, filtragem de imagens e separação de sequências na comunicação multiusuário [18], [19], [20].

Em 1947 Claude Shannon publicou um artigo intitulado "A Mathematical Theory of Communication" [21]. Os sistemas representados pelo modelo clássico, de um sistema de comunicação proposto por Shannon, chamados de sistemas ponto-a-ponto, compreendem

apenas um transmissor e um receptor, que trocam informações por meio de um canal ruidoso, isto é, um canal que pode modificar aleatoriamente as mensagens transmitidas. Entretanto, há casos em que vários usuários desejam se comunicar com um mesmo destinatário fazendo uso simultâneo de um mesmo canal de comunicações. Exemplos de acesso múltiplo são a transmissão de vários canais de voz por meio de um único cabo coaxial em uma rede telefônica, o *up-link* em um sistema de comunicação via satélite e a troca de mensagens e informações entre um computador central e seus vários terminais em uma rede de computadores. Sistemas de comunicação de acesso múltiplo foram estudados primeiramente por Shannon em 1961 [22] e desde então têm sido investigado por diversos autores [23]-[29].

Em 1964 Kautz e Singleton [30] apresentaram um modelo matemático de sistema de transmissão no qual mais de um usuário pode acessá-lo simultaneamente, com a saída deste sendo a soma *booleana* dos sinais enviados pelos usuários. Chien e Frazer [31] introduziram o conceito de superposição de códigos utilizando a adição módulo 2. O canal binário aditivo de dois usuários (2-BAC, do inglês *two-user Binary adder Channel*) [28] é um modelo de canal de comunicação bastante conhecido. Em [32], foi proposto um modelo de canal somador real (RAC, do inglês *real adder channel*) em que, pela primeira vez, autossequências de uma transformada discreta, a de Fourier, foram utilizadas para prover o acesso simultâneo de até 4 usuários. Os modelos de transmissão citados acima são chamados, em geral, de canais de acesso múltiplo com T usuários. No presente trabalho, a transformada numérica de Fourier (TNF), é usada como base para um sistema de múltiplo acesso.

A ideia original de esquemas desse tipo, baseados em transformadas discretas, foi inicialmente proposta por Campello de Souza e de Oliveira [32]. Uma nova técnica de comunicação multiusuário foi proposta, baseada na autoestrutura da DFT [33]. Especificamente um canal real aditivo livre de ruído é considerado e as assinaturas de usuários são autossequências da matriz da DFT; autovalores distintos são associados para cada usuário e, após a adição pelo canal, as correspondentes autossequências são separadas através da solução de um sistema de equações lineares. A autoestrutura das transformadas discretas foi investigada pela primeira vez por Parks e McClellan, que consideraram a DFT [33]. No referido trabalho, foram determinadas as formas dos autovetores da DFT e as multiplicidades dos seus respectivos autovalores. Posteriormente, outros trabalhos realizaram uma análise semelhante, considerando alguns tipos de transformadas trigonométricas [34],[35],[36]. Com

base na autoestrutura das transformadas trigonométricas discretas, foi apresentado por Lima e Campello de Souza um método de múltiplo acesso ao RAC, em que cada usuário transmite sua informação através de autossequências de uma DTT [37]. Cavalcanti e Campello de Souza [38] investigaram as autossequências da TNF e um sistema de múltiplo acesso para o RAC com 4 usuários, utilizando as autossequências da TNF como assinaturas, foi proposto.

As FFTTs, recentemente introduzidas, tiveram sua autoestrutura analisada e aplicada ao cenário de comunicação multiusuário [20],[17],[39]. Em um trabalho recente, Lima e Campello de Souza apresentaram a idéia da comunicação multiusuário baseada na transformada discreta fracional de Fourier (DFrFT) [40].

Nesta dissertação, mostra-se como a idéia de múltiplo acesso ao canal $GF(p)$ aditivo pode ser implementada utilizando as autossequências da TNF, no qual a adição é realizada sobre $GF(p)$. Em particular, é dada ênfase ao caso em que dois usuários transmitem simultaneamente em um canal $GF(p)$ aditivo para um único receptor. Este canal é denominado de (2-GAC, do inglês "*Galois Adder Channel*") e realiza a adição módulo p dos sinais enviados pelos dois usuários. A técnica consiste, basicamente, em utilizar autossequências que podem ser "misturadas" num mesmo canal e, posteriormente, recuperadas. Sob este aspecto, a presente técnica pode ser comparada a uma espécie de DS-CDMA (do inglês *Direct Sequence-Code Division Multiple Access*), em que as sequências de espalhamento seriam os autovetores da matriz de transformação considerada [41].

No contexto de códigos corretores de erros, a FFFT pode ser utilizada, por exemplo, para descrever códigos de bloco [1]. Neste caso, o problema da codificação para controle de erros em sistemas de comunicação digital é analisado sob o prisma da FFFT definida apropriadamente como um mapeamento relacionando vetores com componentes em $GF(p^m)$, $m > 1$[42]. Novas famílias de códigos corretores, baseados na autoestrutura da TNF unitária, chamados códigos de Fourier, foram introduzidos em [43], [44]. A matriz de verificação de paridade, a dimensão e a distância mínima de Hamming do código são obtidas e uma técnica de decodificação baseada na autoestrutura da TNF unitária, para corrigir um e dois erros, é proposta.

O objetivo principal deste trabalho é implementar por meio de simulação computacional o sistema de múltiplo acesso ao canal $GF(p)$ aditivo, em presença de ruído gaussiano branco aditivo (RGBA), a fim de avaliar um aspecto prático do esquema proposto, em particular, a

análise do desempenho da técnica em canais ruidosos. De fato, o desempenho de tal sistema, que pode ser medido pela taxa de símbolos interpretados com erro, depende da relação sinal-ruído ao qual o mesmo está submetido. Naturalmente, o uso de códigos corretores de erro contribui para a elevação do desempenho. No presente trabalho isto é feito com o uso de codificação de canal baseada nos códigos de Fourier.

1.1 CONTEÚDO DA DISSERTAÇÃO

Após esta introdução, o Capítulo 2 inicia-se com uma revisão das transformadas numéricas de Fourier, incluindo sua definição e algumas propriedades. Este Capítulo descreve a autoestrutura da TNF, mostrando detalhadamente como as autossequências dessas transformadas são geradas. São ainda apresentadas as transformadas numéricas de Fermat e de Mersenne [45], cujas implementações, em muitos casos, não requerem multiplicações.

No Capítulo 3, os códigos baseados na autoestrutura da TNF unitária introduzidos recentemente por Freire, et al [43], [44], chamados códigos de Fourier, têm o processo de sua construção descrito, sendo apresentados os algoritmos para a correção de um e dois erros com os mesmos.

No Capítulo 4, o sistema de comunicação multiusuário sobre o canal $GF(p)$ aditivo tem suas principais características discutidas e as ideias básicas para a concepção de um esquema de comunicação com dois usuários são explicadas em mais detalhes. Em seguida, discutem-se esquemas hierárquicos [20],[39], e o uso de transformadas cuja matriz de transformação possui um número arbitrário de autovalores distintos, propostos por Lima e Campello de Souza, para concepção de sistemas de acesso múltiplo com número máximo de usuários simultâneos arbitrário [40].

O Capítulo 5 descreve a implementação por meio de uma simulação do sistema de múltiplo acesso para o 2-GAC a fim de avaliar o desempenho do esquema proposto considerando um canal com ruído. Os resultados conseguidos pelo mesmo com e sem codificação de canal são apresentados. Finalmente, no Capítulo 6, são tecidos os comentários finais e são apontadas as sugestões de trabalhos futuros.

O Apêndice A mostra duas tabelas, com as palavras-código utilizadas pelos usuários 1 e 2.

O Apêndice B mostra a tabela com todos os caracteres (ASCII, do inglês "*American Standard Code for Information Interchange*") imprimíveis, utilizados como informação dos usuários, e seus respectivos valores correspondentes em decimal.

O Apêndice C contém as principais rotinas computacionais necessárias para criar as funções individuais dos blocos *Embedded MATLAB function*, utilizados extensivamente na composição do simulador e que tornaram possível a execução do programa no ambiente do *Simulink*.

CAPÍTULO 2
A TRANSFORMADA NUMÉRICA DE FOURIER

Uma transformação semelhante à transformada discreta de Fourier pode ser definida em um corpo finito e pode ser calculada de forma eficiente pelos algoritmos FFT (do inglês, *fast Fourier transform*). A transformada de Fourier sobre corpos finitos (TFCF), como ficou conhecida, foi introduzida por J. M. Pollard em 1971 [4].

Este capítulo apresenta uma breve revisão acerca da principal ferramenta matemática usada nesta dissertação, a transformada numérica de Fourier (TNF). A autoestrutura da TNF é examinada e algumas famílias especiais de interesse prático dessas transformadas são apresentadas.

2.1 TRANSFORMADAS NUMÉRICAS

É um fato bem conhecido na literatura que o conjunto dos inteiros módulo *p*, $Z_p = \{0,1,...., p-1\}$, equipado com as operações de adição e multiplicação módulo *p*, é uma estrutura algébrica denominada corpo finito [45]. As transformadas de corpo finito consideradas neste trabalho são definidas sobre este corpo, que aqui é denotado por *GF*(*p*) (*GF*, do inglês *Galois Field*; as expressões campo de *Galois* e corpo finito têm o mesmo significado). O número de elementos de um corpo finito, denotado por *q*, é uma potência de um número primo, isto é, $q = p^r, r \geq 1$ inteiro. Quando $r > 1,$ os elementos de *GF*(*q*) são todos os polinômios de grau (não negativo) menor ou igual a $(r-1)$, com coeficientes em *GF*(*p*), e a aritmética da estrutura é definida como sendo a de adição e multiplicação módulo um polinômio irredutível de grau r, com coeficientes em *GF*(*p*). Transformadas de corpo finito definidas sobre *GF*(p^r), $r > 1,$ que têm aplicações em Processamento Digital de Sinais, Codificação de Canal e Criptografia, não são consideradas nesta dissertação [40],[47].

Definição 2.1: Seja $x = (x_0, x_1, ..., x_{N-1})$ um vetor de comprimento *N* e com componentes em *GF*(*q*), em que $q = p^r$. O vetor $X = (X_0, X_1, ..., X_{N-1})$, com componentes em *GF*(q^m) dadas por

$$X_k = \sum_{n=0}^{N-1} x_n \alpha^{kn}, \tag{2.1}$$

em que α é um elemento de ordem *N* em *GF*(q^m), é dito ser a transformada de Fourier de corpo finito (TFCF) de *x*. Quando $r = m = 1$, a transformação mapeia vetores com componentes em *GF*(*p*) e é chamada Transformada Numérica de Fourier (TNF). O elemento α é denominado núcleo da transformada. ■

Observe que a TNF está restrita àqueles comprimentos *N* para os quais existe um elemento α de ordem *N* em GF(*p*), ou seja, *N* precisa ser um divisor de (*p* - 1).

Definição 2.2: As sequências $x = (x_0, x_1, ..., x_{N-1})$ e $X = (X_0, X_1, ..., X_{N-1})$, de elementos de *GF*(*p*), formam um par da TNF unitária se

$$X_k = (\sqrt{N})^{-1} (\mathrm{mod}\, p) \sum_{n=0}^{N-1} x_n \alpha^{kn} \tag{2.2}$$

e

$$x_n = (\sqrt{N})^{-1} (\mathrm{mod}\, p) \sum_{k=0}^{N-1} X_k \alpha^{-kn}, \tag{2.3}$$

em que α é um elemento de ordem multiplicativa *N* em *GF*(*p*) e todos os cálculos são efetuados módulo *p*; o fator de escalonamento $\sqrt{1/N} (\mathrm{mod}\, p)$, cuja condição de existência é que *N* seja um resíduo quadrático sobre *GF*(*p*), tem a função de tornar a transformada unitária. ■

O par TNF resultante é denotado por $x \leftrightarrow X$ ou $X = Fx$ em que, a partir da Definição 2.2, a matriz de transformação, F, é dada por

$$F = (\sqrt{N})^{-1} \, (mod\ p) \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{N-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{N-1} & \alpha^{2(N-1)} & \cdots & \alpha^{(N-1)(N-1)} \end{pmatrix}.$$

2.2 AUTOSSEQUÊNCIAS DA TRANSFORMADA NUMÉRICA DE FOURIER

Definição 2.3: Se a TNF da sequência x satisfaz $X = \lambda x$, em que $\lambda \in GF(p^2)$, então x é dito ser uma autossequência da TNF, ou em forma de operador, $\Gamma(x) = \lambda x$, em que Γ denota o operador TNF unitário e λ é chamado o autovalor associado a x. ■

É possível provar que [48], se p = 4k+3 então $j \in GF(p^2)$, k inteiro, e se p = 4k+1, então $j = \left(\frac{p-1}{2}\right)!(mod\ p)$.

Os autovalores e autovetores da TNF são caracterizados nos lemas a seguir [33].

Lema 2.1: i) Os autovalores da TNF unitária são as quatro raízes da unidade, $(\pm 1, \pm j)$, em que $j^2 \equiv -1(mod\ p)$. ■

ii) Se x é uma autossequência da TNF, então ela tem simetria par (*i.e.*, $x_i \equiv x_{N-i}(mod\ p)$), se $\lambda \equiv \pm 1(mod\ p)$ e simetria ímpar (*i.e.*, $x_i \equiv -x_{N-i}(mod\ p)$), se $\lambda \equiv \pm j(mod\ p)$. ■

Sequências pares e ímpares podem ser utilizadas para construir autossequências da TNF [32].

Lema 2.2: Seja $x \leftrightarrow X$ um par TNF. As sequências $y = E(x) \pm E(X)$ são autossequências com autovalores associados, respectivamente, $\lambda \equiv \pm 1 (\text{mod } p)$, em que o operador $E(x)$ denota a parte par de x. ■

Corolário 2.1: Toda sequência x, de simetria par, gera uma autossequência $y = x \pm X$. ■

Lema 2.3: Seja $x \leftrightarrow X$ um par TNF. As sequências $y = O(x) \mp jO(X)$ são autossequências com autovalores associados $\lambda \equiv \pm j (mod\ p)$, respectivamente, em que o operador $O(x)$ denota a parte ímpar de x. ■

Corolário 2.2: Toda sequência x, de simetria ímpar, gera uma autossequência $y = x \mp jX$. ■

Lema 2.4: Se $p \equiv 1 (mod\ 4)$, os autovalores da TNF são reais, isto é, $\lambda \in GF(p)$ [48]. ■

Exemplo 2.1: Considere a sequência par x = (5 1 1 1 1) com valores sobre $GF(61)$ e a matriz TNF (5 x 5) sobre $GF(61)$, com $\alpha = 9$,

$$F = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 \\ 7 & 2 & 18 & 40 & 55 \\ 7 & 18 & 55 & 2 & 40 \\ 7 & 40 & 2 & 55 & 18 \\ 7 & 55 & 40 & 18 & 2 \end{pmatrix}.$$

A TNF de x é X = (2 28 28 28 28). Como x é uma sequência par, de acordo com o Corolário 2.1, a sequência $y = x + X$ = (7 29 29 29 29) é uma autossequência com autovalor associado $\lambda \equiv 1 \text{ (mod } 61)$. ■

Exemplo 2.2: Considere a sequência ímpar x = (0 60 14 47 1) e a mesma matriz TNF do exemplo anterior. O espectro de x é X = (0 50 32 29 11). Como x é uma sequência ímpar, de acordo com o Corolário 2.2, a sequência $y_2 = x - jX$ = (0 59 28 33 2) é uma autossequência com autovalor associado $\lambda \equiv j = 11 (\text{mod } 61)$. ■

2.3 PROPRIEDADES DA TNF

Sejam $x = (x_n) \leftrightarrow X = (X_k)$ e $y = (y_n) \leftrightarrow Y = (Y_k)$, $n, k = 0, 1,..., N-1$, pares da TNF unitária de comprimento N. Algumas propriedades úteis da TNF são mostradas a seguir, em que os índices devem ser considerados módulo N [49].

P1. **Linearidade**

$ax + by \leftrightarrow aX + bY$

P2. **Deslocamento no tempo**

Se $y_n = x_{n-s}$, s um inteiro fixo, então $Y_k = \alpha^{ks} X_k$.

P3. **Reversão no tempo**

Se $y_n = x_{-n,}$ em que $x_{-n} = x_{N-n,}$ então $Y_k = X_{-k}$.

P4. **Convolução cíclica**

Se $z_n = (x_n) *_N (y_n)$, em que $*_N$ denota convolução cíclica $_{N\text{x}N}$, então $Z_k = (X_k)(Y_k)$,[45].

2.4 A TRANSFORMADA NUMÉRICA DE FOURIER-FERMAT

A complexidade computacional para se implementar uma transformada discreta é medida pelo número de operações necessárias para sua computação. Esta complexidade também é denominada complexidade aritmética, uma vez que, neste cenário, as operações envolvidas são a multiplicação e a adição. Os chamados algoritmos rápidos para cálculo de transformadas discretas geralmente procuram reduzir a complexidade multiplicativa (dada pelo número de multiplicações), uma vez que esta é a operação de maior custo [45].

Entre algumas considerações práticas importantes, no que diz respeito à eficiência computacional da implementação de uma transformada numérica, pode-se destacar a escolha do núcleo α da transformada (note que, dado o comprimento N da transformada, essa escolha não existe no caso da DFT usual, sobre o corpo dos números reais). Uma escolha atrativa para α é a de uma potência de dois, isto é, $\alpha = 2^s$, para algum s inteiro positivo, já que os sistemas eletrônicos digitais são baseados em operações binárias em que as multiplicações por potências

de dois são deslocamentos. Nesse contexto, uma aritmética módulo p adequada corresponde àquela em que p é um número primo de Fermat, isto é, um primo da forma $p = 2^m + 1$ [45].

Um inteiro da forma $F_n = 2^{2^n} + 1$, $n \geq 0$, é chamado número de Fermat por causa do matemático francês Pierre de Fermat (1601-1665), que conjecturou que os números dessa forma eram primos. Os números de Fermat que são primos são chamados primos de Fermat. Sabe-se que, se $2^m + 1$ é um número primo, então m é uma potência de dois. No entanto, a recíproca não é verdadeira uma vez que, por exemplo, $2^{32} + 1$ não é primo. Os únicos primos de Fermat conhecidos até hoje são F_0, F_1, F_2, F_3 e F_4, respectivamente, 3, 5, 17, 257 e 65537 [48].

Nesta seção, as transformadas numéricas definidas sobre GF(p), em que p é um primo de Fermat, são consideradas. Tais transformadas, denominadas de transformadas numéricas de Fourier-Fermat (TNFF), permitem implementações com complexidade multiplicativa nula. Em $GF(p)$, quando p é um primo de Fermat, qualquer fator de $(p - 1)$ é uma potência de 2. Nesse caso, a TNFF

$$X_k = \sum_{n=0}^{N-1} x_n \alpha^{kn} (\mathrm{mod}\, p)$$

existe sempre que N é um divisor de 2^m e α é um elemento de ordem N. Uma vez que o comprimento N da transformada é uma potência de dois, a mesma pode ser calculada, por meio da FFT de Cooley-Tukey de base 2, com apenas $(N/2)\log_2 N$ multiplicações e $(N)\log_2 N$ adições [45]. Desde que os elementos da sequência que será transformada são multiplicados por potências do núcleo, a complexidade multiplicativa do cálculo da transformada depende fortemente da escolha de α.

Utilizando-se aritmética binária, multiplicações por uma potência de dois podem ser implementadas como deslocamentos. Pela congruência $2^{2m} \equiv (-1)^2 \equiv 1$ (mod $2^m + 1$), o inteiro $\alpha = 2$ tem ordem $2m$, módulo $(2^m + 1)$, e, portanto, pode ser usado como núcleo de uma TNFF de comprimento $2m$.

Exemplo 2.3: A expressão

$$X_k = \sum_{n=0}^{31} x_n 2^{kn} \pmod{2^{16}+1},$$

k = 0, 1,..,31, define uma TNFF cuja computação requer apenas adições e deslocamentos. Tal transformada é dita ser livre de multiplicações. ■

Um núcleo adequado para se obter uma transformada de comprimento 4m é $\alpha = \sqrt{2}$. É possível mostrar que $\sqrt{2} \equiv 2^{m/4}(2^{m/2} - 1) \pmod{2^m+1}$. Assim, toda potência de $\sqrt{2}$ tem a forma $(2^a \pm 2^b)$ [45].

Exemplo 2.4: Uma TNFF de comprimento 64, sem multiplicações, em $GF(2^{16}+1)$, tem a forma

$$X_k = \sum_{n=0}^{63} x_n (2^{12} - 2^4)^{kn},$$

k = 0, 1,...,63. Quando esta transformada é calculada com qualquer algoritmo, todas as multiplicações são por uma constante da forma $2^a \pm 2^b$, que podem ser implementadas como um par de deslocamentos e uma adição ou subtração. ■

2.5 A TRANSFORMADA NUMÉRICA DE FOURIER-MERSENNE

Um conjunto de números inteiros de particular interesse é o conjunto dos números de Mersenne, que são números da forma $2^m - 1$, $m > 1$, denominados assim por causa do Monge francês Marin Mersenne (1588 - 1644), que fez uma conjectura parcialmente incorreta, porém provocativa sobre sua primalidade. Os números de Mersenne que são primos são chamados primos de Mersenne. A conjectura afirmava que M_p é primo para p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257 e composto para todos os outros primos $p < 257$. A definição sobre a veracidade dessa conjectura levou cerca de 300 anos e apenas em 1947 é que o exame da primalidade de M_p para os 55 primos na faixa $p \leq 257$ foi concluído [48]. Dentre os primos de Mersenne, que são denotados por M_m, encontram-se os maiores números primos conhecidos. O maior número primo conhecido até o momento (agosto 2011) é $M_{43112609} = 2^{43112609} - 1$, um primo de

Mersenne com 12.978.189 dígitos, descoberto em 23 de agosto de 2008 [46]. A Tabela 2.1 mostra os oito primeiros primos de Mersenne e a correspondente decomposição de M_m -1 em fatores primos.

Se 2^m –1 é um número primo, então o campo de Galois $GF(2^m - 1)$ existe. Em todo corpo finito $GF(p)$, existe uma TNF de comprimento N, para todo N que divide (p-1). Portanto, existe uma TNF de comprimento N, sobre $GF(2^m - 1)$, sempre que N divide $(2^m - 2)$. Estas transformadas são chamadas transformadas numéricas de Fourier-Mersenne (TNFM) e são atraentes para aplicações em Engenharia pois a operação de multiplicação é mais simples nos corpos $GF(p)$, em que p é um primo de Mersenne. Mais especificamente, se os inteiros são representados como números binários de m-bits, como $2^m \equiv 1 (mod\ 2^m - 1)$, a complexidade da execução das operações é igual à complexidade da aritmética complemento de um [45].

Tabela 2.1 – *Os primeiros oito números primos de Mersenne*

m	$M_m = 2^m - 1$	$M_m - 1 = 2(2^{m-1} - 1)$
2	**3**	**2**
3	**7**	**2 · 3**
5	**31**	**2 · 3 · 5**
7	**127**	**2 · 32 · 7**
13	**8191**	**2 · 32 · 5 · 7 · 13**
17	**131.071**	**2 · 3 · 5 · 17 · 257**
19	**524.287**	**2 · 33 · 7 · 19 · 73**
31	**2147.483.647**	**2 · 32 · 7 · 11 · 31 · 151 · 331**

O comprimento N da TNFM é um divisor de $M_m - 1 = 2^m - 2$. Portanto, estas transformadas não podem ser calculadas pela FFT de Cooley-Tukey de base 2, uma vez que N não é uma potência de dois.

Exemplo 2.5: Os comprimentos da TNFM que podem ser construídas sobre $GF(2^{13} - 1)$ são aqueles valores N que dividem $2^{13} - 2$. Existem 48 escolhas para o valor de N, uma vez que a fatoração canônica deste inteiro é $2^{13} - 2 = 2 \cdot 5 \cdot 7 \cdot 9 \cdot 13$. Uma possibilidade é escolher o

elemento $\alpha = -2$ para núcleo da transformada, pois, como $2^{13} \equiv 1 \pmod{2^{13}-1}$, -2 tem ordem 26. Portanto a expressão da TNFM obtida é

$$X_k = \sum_{n=0}^{25} x_n (-2)^{kn},$$

$k = 0,..., 25$. Na expressão acima, todas as multiplicações são multiplicações por potências de 2 e assim esta transformada apresenta complexidade multiplicativa nula, podendo ser computada apenas com adições e deslocamentos.

CAPÍTULO 3
OS CÓDIGOS DE FOURIER

Uma nova família de códigos de bloco lineares multiníveis foi introduzida em [43], [44]. Estes códigos foram denominados códigos de Fourier, por terem sua construção baseada na autoestrutura da TNF. Especificamente, as palavras-código de um código de Fourier são autossequências da TNF, associadas a um dos seus quatro autovalores possíveis. Um código de Fourier de comprimento n, dimensão k e distância mínima d, associado ao autovalor λ, é denotado por $F^{\lambda}(n, k, d)$.

Neste trabalho, os códigos de Fourier são empregados para combater os efeitos do ruído no sistema de comunicação multiusuário proposto no Capítulo 4, quando o mesmo é usado em um canal de transmissão submetido a ruído gaussiano branco aditivo.

3.1 A CONSTRUÇÃO DOS CÓDIGOS DE FOURIER

Considere a matriz quadrada $F_{(N\times N)}$ de ordem N da TNF dada por

$$F = (\sqrt{N})^{-1} \pmod{p} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{N-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{N-1} & \alpha^{2(N-1)} & \dots & \alpha^{(N-1)(N-1)} \end{pmatrix},$$

em que $\alpha \in GF(p)$ tem ordem multiplicativa N. Se x é uma autossequência da transformação linear F e λ é um autovalor associado a x, então $Fx = \lambda x$. Observe que, se $\mathbf{I}$ for a matriz identidade de ordem n, então a expressão anterior pode ser escrita na forma $Fx = (\lambda\mathbf{I})\, x$, ou ainda, $(F - \lambda\mathbf{I})\, x = 0$. Dessa forma, a matriz $H^{(\lambda)} = (F - \lambda I)$ desempenha um papel análogo ao da matriz de verificação de paridade de um código de bloco linear $C(n, k, d)$ [47], em que $n - k = posto\ (F - \lambda I)$. Assim, associado a cada um dos quatro autovalores da TNF, existe um código de bloco linear sobre GF(p), o código de Fourier $F^{\lambda}(n, k, d)$.

No que se segue, as matrizes de paridade e geradora do código são apresentadas na forma padrão $H^{(\lambda)} = [\, I_{n-k} \,|\, P]$ e $G^{(\lambda)} = [-P^T | \, I_k]$, respectivamente.

Exemplo 3.1: Códigos de Fourier de comprimento 7 sobre *GF*(29). Considerando α = 7, um elemento de ordem 7 no corpo dado, em que $\sqrt{7} \equiv 23 (\mathrm{mod} 29)$, obtém-se, a partir da matriz *F*,

$$F - \lambda I = \begin{pmatrix} 24-\lambda & 24 & 24 & 24 & 24 & 24 & 24 \\ 24 & 23-\lambda & 16 & 25 & 1 & 7 & 20 \\ 24 & 16 & 1-\lambda & 20 & 23 & 25 & 7 \\ 24 & 25 & 20 & 16-\lambda & 7 & 23 & 1 \\ 24 & 1 & 23 & 7 & 16-\lambda & 20 & 25 \\ 24 & 7 & 25 & 23 & 20 & 1-\lambda & 16 \\ 24 & 20 & 7 & 1 & 25 & 16 & 23-\lambda \end{pmatrix}.$$

Em forma escalonada padrão, as matrizes de verificação de paridade, considerando os quatro autovalores λ = ±1, ±*j*, são, respectivamente,

$$H^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 10 & 24 \\ 0 & 1 & 0 & 0 & 0 & 0 & 28 \\ 0 & 0 & 1 & 0 & 0 & 28 & 0 \\ 0 & 0 & 0 & 1 & 0 & 24 & 4 \\ 0 & 0 & 0 & 0 & 1 & 24 & 4 \end{pmatrix}; \qquad H^{(-1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 13 & 9 \\ 0 & 1 & 0 & 0 & 0 & 0 & 28 \\ 0 & 0 & 1 & 0 & 0 & 28 & 0 \\ 0 & 0 & 0 & 1 & 0 & 19 & 9 \\ 0 & 0 & 0 & 0 & 1 & 19 & 9 \end{pmatrix};$$

$$H^{(j)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 18 \\ 0 & 0 & 0 & 1 & 0 & 0 & 19 \\ 0 & 0 & 0 & 0 & 1 & 1 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 11 \end{pmatrix}; \qquad H^{(-j)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 25 & 3 \\ 0 & 0 & 0 & 0 & 1 & 4 & 26 \end{pmatrix}.$$

Portanto, têm-se os quatro códigos de Fourier

$F^{\,1}[7, 2, 5]$, gerado por $G^{(1)} = \begin{pmatrix} 19 & 0 & 1 & 5 & 5 & 1 & 0 \\ 5 & 1 & 0 & 25 & 25 & 0 & 1 \end{pmatrix}$

F^{-1} [7, 2, 5], gerado por $G^{(-1)} = \begin{pmatrix} 16 & 0 & 1 & 10 & 10 & 1 & 0 \\ 20 & 1 & 0 & 20 & 20 & 0 & 1 \end{pmatrix}$

F^{j} [7, 1, 5], gerado por $G^{(j)} = (0 \quad 28 \quad 11 \quad 10 \quad 19 \quad 18 \quad 1)$

F^{-j} [7, 2, 5], gerado por $G^{(-j)} = \begin{pmatrix} 0 & 0 & 28 & 4 & 25 & 1 & 0 \\ 0 & 28 & 0 & 26 & 3 & 0 & 1 \end{pmatrix}$

3.2 PARÂMETROS DOS CÓDIGOS DE FOURIER

O comprimento n do código de Fourier $F^{\lambda}(n, k, d)$ é a dimensão N da matriz TNF unitária. A dimensão k é a multiplicidade geométrica do autovalor λ, ou seja, é a dimensão do subespaço de autossequências associadas a λ. A tabela 3.1 mostra as multiplicidades dos quatro autovalores da TNF [33]. Uma cota superior para a distância mínima d é demonstrada em [44]. A distância mínima de Hamming de um código de Fourier $F^{\lambda}(n, k, d)$ satisfaz $d \leq n - 2k + 2$.

Tabela 3.1 - *Multiplicidade dos Autovalores da TNF unitária*.

N	$\lambda = 1$	$\lambda = -1$	$\lambda = j$	$\lambda = -j$
$4m$	$m + 1$	m	m	$m - 1$
$4m + 1$	$m + 1$	m	m	m
$4m + 2$	$m + 1$	$m + 1$	m	m
$4m + 3$	$m + 1$	$m + 1$	$m + 1$	m

3.3 DECODIFICAÇÃO DE CÓDIGOS DE FOURIER E DETECÇÃO DE ERRO

As estruturas adicionais que estes códigos apresentam são obtidas das autossequências da transformada numérica de Fourier, o que permitiu a criação de algoritmos de decodificação simples. A correção de erros opera de modo que, primeiro, aplica-se o algoritmo para correção de um único erro e, se a decodificação não for possível, a ocorrência de dois erros é assumida. Seja qual for o caso, o algoritmo de decodificação depende da sequência recebida ter, ou não,

simetria. Um aspecto interessante destes códigos é o uso de uma transformada rápida para calcular a síndrome *S*.

3.3.1 CÁLCULO DA SÍNDROME

Considerando que a palavra-código (autossequência da TNF) $x \in F^{\lambda}(n, k, d)$ é transmitida por um canal ruidoso, é assumido que a sequência recebida, $r = (r_0,...,r_{N-1})^T$, tem componentes dadas por

$$r_i = x_i + e_i \,(\mathrm{mod}\, p),$$

em que $e = (e_0, e_1, e_2 ,.., e_{N-1})^T$ denota a sequência erro (ruído aditivo), possivelmente introduzida pelo canal durante a transmissão. Uma vez que qualquer palavra-código deve obedecer à condição $(F-\lambda I)x = 0$, um mecanismo de detecção de erros é implementado com base na expressão acima, que adota a forma seguinte: o vetor $S \triangleq (F- \lambda I)r$ é chamado a síndrome da sequência recebida. Desde que a sequência recebida seja uma autossequência, então *S* é zero. Se *S* tem um valor não-zero, então o procedimento detecta uma sequência que não pertence ao código, o que significa a ocorrência de erros durante a transmissão. Similares aos códigos cíclicos (CRC, do inglês Cyclic Redundancy Checking), por permitir um cálculo rápido da síndrome, a adaptação de tais códigos (códigos de Fourier) em protocolos com ARQ-Híbrido é, certamente, um tópico interessante para investigações futuras.

3.3.2 CORREÇÃO DE UM ÚNICO ERRO (CASO SIMÉTRICO)

A autoestrutura da TNF é a base dos algoritmos de decodificação dos códigos de Fourier. Um deles é o algoritmo para correção de um único erro. Esta decodificação é baseada na relação de simetria das autossequências da TNF e na possível simetria da palavra recebida. Considere $r = (r_0, r_1, r_2 ,..., r_{N-1})$ a sequência recebida. No que se segue, sem perda de generalidade, *N* é considerado ímpar e $\lambda = \pm 1$.

Na ocorrência de um único erro, caso *r* apresente o mesmo tipo de simetria das autossequências associadas a λ, este deve ter ocorrido na primeira posição do vetor *r*

(correspondente ao símbolo r_0); então o procedimento de correção é recalcular r_0 de acordo com a Definição 2.3 e substituí-lo na sequência recebida. Portanto, para decodificar a sequência recebida, usa-se o seguinte algoritmo:

1. Mudar o valor de r_0 recebido para $r_0 = (\lambda\sqrt{N} -1)^{-1}(r_1 + r_2 +...+ r_{N-1})$;

2. Se a nova sequência r obtida é uma autossequência, a decodificação está completa. Caso contrário, mais de um erro ocorreu. Se este for o caso, o algoritmo para correção de dois erros, no caso simétrico, procura em qual par de símbolos ocorreram os erros (r_i, r_{N-i}), $i \neq 0$, conforme descrito a seguir.

3.3.3 CORREÇÃO DE DOIS ERROS (CASO SIMÉTRICO)

1. Para $1 \leq i \leq (N-1) / 2$, fazer $r_i = 1/2 \left(\lambda\sqrt{N}\, r_0 - \sum_{\substack{j=0 \\ j \neq i, N-i}}^{N-1} r_j \right)$ e $r_{N-i} = r_i$.

2. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, mais de dois erros ocorreram.

3.3.4 CORREÇÃO DE UM ÚNICO ERRO (CASO NÃO-SIMÉTRICO)

No caso não simétrico, N ímpar e $\lambda = \pm 1$ também são considerados. Se os símbolos r_i e r_{N-i} da sequência recebida são diferentes, o único erro ocorreu na posição r_i ou na posição r_{N-i}.

O algoritmo de decodificação para esse caso é:

1. Substituir r_i por r_{N-i} ;

2. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, então substituir r_{N-i} por r_i;

3. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, mais de um erro ocorreu.

3.3.5 CORREÇÃO DE DOIS ERROS (CASO NÃO-SIMÉTRICO)

Se dois erros ocorreram, são três as opções:

i) Erros nos símbolos r_0 e r_i, $i \neq 0$.

Algoritmo de Decodificação 1:

1. Substituir r_i por r_{N-i};

2. Fazer $r_0 = (\lambda\sqrt{N} -1)^{-1}(r_1 + r_2 +...+ r_{N-1})$;

3. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, substitituir r_{N-i} por r_i e fazer $r_0 = (\lambda\sqrt{N} -1)^{-1}(r_1 + r_2 +...+ r_{N-1})$;

4. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, usar o Algoritmo de Decodificação 2.

ii) Um erro em r_i e outro em r_{N-i}. O algoritmo de decodificação é parecido com o do caso simétrico, com a diferença de que agora as posições dos erros são conhecidas.

Algoritmo de Decodificação 2:

1. Fazer $r_i = 1/2\ (\lambda\sqrt{N}\, r_0 - \sum_{\substack{j=0 \\ j \neq i,N}}^{N-1} r_j)$ e $r_{N-i} = r_i$.

2. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, mais de dois erros ocorreram.

iii) Um erro em r_i, $i \neq 0$, e outro em r_j, $j \neq N - i$. Nesse caso, é necessário fazer pelo menos quatro substituições para satisfazer a simetria e verificar a condição de autossequência.
Algoritmo de Decodificação 3:

1. Substituir r_i por r_{N-i} e substituir r_j por r_{N-j};

2. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, substituir r_{N-i} por r_i e substituir r_j por r_{N-j};

3. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, substituir r_i por r_{N-i} e substituir r_{N-j} por r_j;

4. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, substituir r_{N-i} por r_i e substituir r_{N-j} por r_j;

5. Se a sequência obtida é uma autossequência, a decodificação está completa. Caso contrário, mais do que dois erros ocorreram.

CAPÍTULO 4
UM SISTEMA DE ACESSO MÚLTIPLO SOBRE O CANAL GF(p) ADITIVO

Neste Capítulo, um sistema de múltiplo acesso para o canal $GF(p)$ aditivo com dois usuários, que é denominado de 2-GAC, utilizando as autossequências da TNF como assinaturas de usuários, é descrito. Expressões para a recuperação da informação dos usuários sãoobtidas. Aqui, utiliza-se a autoestrutura da TNF para a separação das sequências. Considera-se um canal aditivo sobre corpos finitos livre de erro o qual é compartilhado por diferentes usuários de modo síncrono. O procedimento consiste em associar um autovalor e, portanto, um conjunto de autovetores de determinada TNF a cada usuário. A informação a ser enviada por um usuário é mapeada sobre autovetores. Uma vez que autovetores relacionados a diferentes autovalores são ortogonais, após serem somados pelo canal $GF(p)$ aditivo, os mesmos podem ser recuperados a partir da solução de um sistema de equações lineares. Este esquema é ilustrado nas seções subsequentes.

4.1 O CANAL $GF(p)$ ADITIVO

No presente trabalho, uma utilização das autossequências da TNF como assinaturas de usuários sobre o 2-GAC é sugerida. No que se segue, $x_i[n]$ denota a autossequência do usuário i, cuja TNF é dada por $X_i[k]$, e a operação de adição entre sequências ($x_1[n] + x_2[n]$) representa adição, componente a componente, módulo p, em que $p > 2$. A tabela 4.1 apresenta exemplos de autossequências da TNF para diversos valores de p e de λ.

4.1.1 O 2-GAC

O conjunto de todas as autossequências associadas a um autovalor λ, munido da operação de adição de autossequências, componente a componente, é um subespaço vetorial em relação a $GF(p)$ [50], denotado por V_λ.

Tabela 4.1 - *Autossequências da transformada numérica de Fourier*.

N	λ	p	**AUTOSSEQUÊNCIA**
4	1	29	3,1,1,1
4	-1	17	15,2,2,2
4	$j = 12$	29	0,28,0,1
7	$j = 12$	29	0,3,9,12,17,20,26
7	$-j = 17$	29	0,9,4,2,27,25,20
8	1	17	14,11,15,16,6,16,15,11
8	-1	17	9,16,7,16,4,16,7,16
12	1	13	5,5,4,9,1,1,6,1,1,9,4,5
12	-1	13	7,8,8,8,8,8,8,8,8,8,8,8
12	$j \equiv 5$	13	0,6,2,11,2,10,0,3,11,2,11,7
12	$-j \equiv 8$	13	0,0,4,5,11,11,0,2,2,8,9,0

Consideremos as autossequências da TNF $x_1[n]$ e $x_2[n]$, definidas em V_1 e V_{-1} e associadas aos usuários 1 e 2, respectivamente. O 2-GAC fornece uma nova sequência $y[n] = x_1[n] + x_2[n]$ (mod p) a partir da qual as sequências de usuários individuais podem ser recuperadas. A partir dos pares TNF $x_1[n] \leftrightarrow X_1[k]$, $x_2[n] \leftrightarrow X_2[k]$ e $y[n] \leftrightarrow Y[k]$, pode-se escrever

$$Y[k] = X_1[k] + X_2[k]. \tag{4.1}$$

Desde que $x_1[n]$ e $x_2[n]$ estejam em autoespaços distintos V_1 e V_{-1}, respectivamente, (4.1) é equivalente a

$$Y[k] = x_1[k] - x_2[k], \tag{4.2}$$

de modo que, a partir de $y[n] = x_1[n] + x_2[n]$ e da expressão (4.2), as sequências de usuários são recuperadas a partir das expressões

$$x_1[n] = \frac{y[n]+Y[n]}{2}; \quad (4.3)$$

$$x_2[n] = \frac{y[n]-Y[n]}{2}, \quad (4.4)$$

que, naturalmente, são avaliadas módulo p (note que, como $p > 2$, (2^{-1}) (mod p) existe). Aqui, algoritmos FFT são utilizados para o cálculo de $Y[k]$. Considerando os quatro autovalores, um total de seis sistemas semelhantes podem ser implementados. A Tabela 4.2 mostra, para cada um deles, expressões para calcular $x_1[n]$ e $x_2[n]$.

Tabela 4.2 - *Expressões de recuperação de usuário para o 2-GAC.*

Autoespaços selecionados	**Usuário1**	**Usuário 2**
(V_1, V_{-1})	$\frac{y+Y}{2}$	$\frac{y-Y}{2}$
(V_1, V_j)	$\frac{-y+Y}{1-j}$	$\frac{y-Y}{1-j}$
(V_1, V_{-j})	$\frac{jy+Y}{1+j}$	$\frac{jy-Y}{1+j}$
(V_{-1}, V_j)	$\frac{jy-Y}{1+j}$	$\frac{y+Y}{1+j}$
(V_{-1}, V_{-j})	$\frac{-jy-Y}{1-j}$	$\frac{y+Y}{1-j}$
(V_j, V_{-j})	$\frac{y-jY}{2}$	$\frac{y+jY}{2}$

4.1.2 O 3-GAC

O GAC com 3-usuários fornece a saída y[n] = x_1[n] + x_2[n] + x_3[n] (*mod p*). Os quatro possíveis autoespaços dados para escolhas de projeto correspondem aos seguintes conjuntos de autovalores: {+1, -1, +j}, {+1, -1, -j}, {+1, +j, -j}, e {-1, +j, -j}. Tomando como exemplo o

projeto associado ao conjunto de autovalores {+1, -1, +j}, e aplicando duas vezes a TNF para y, o conjunto das equações a seguir é obtido:

$$x_1[n] + x_2[n] + x_3[n] \,(mod\ p) = y[n], \tag{4.5}$$

$$x_1[n] - x_2[n] + jx_3[n] \ (mod\ p) = \Gamma(y)\ [n], \tag{4.6}$$

$$x_1[n] + x_2[n] - x_3\,[n] \,(mod\ p) = \Gamma^{(2)}\,(y)\ [n], \tag{4.7}$$

em que o canal é livre de erro. A solução do sistema é expressa por

$$x_1[n] = \frac{E\{y\}+jO\{y\}+Y}{2}, \tag{4.8}$$

$$x_2[n] = \frac{E\{y\}+jO\{y\}-Y}{2}, \tag{4.9}$$

$$x_3[n] = O\{y\}. \tag{4.10}$$

Considerando os demais conjuntos de autovalores, as expressões para recuperar os usuários são indicadas na Tabela 4.3.

Tabela 4.3 - *Expressões de recuperação de usuário para o 3-GAC.*

Autoespaços selecionados	**Usuário 1**	**Usuário 2**	**Usuário 3**
(V_1, V_{-1}, V_j)	$\frac{E\{y\}-jO\{y\}+Y}{2}$	$\frac{E\{y\}+jO\{y\}-Y}{2}$	$O\{y\}$
(V_1, V_{-1}, V_{-j})	$\frac{E\{y\}+jO\{y\}+Y}{2}$	$\frac{E\{y\}-jO\{y\}-Y}{2}$	$O\{y\}$
(V_1, V_j, V_{-j})	$E\{y\}$	$\frac{O\{y\}+jE\{y\}-jY}{2}$	$\frac{O\{y\}-jE\{y\}+jY}{2}$
(V_{-1}, V_j, V_{-j})	$E\{y\}$	$\frac{O\{y\}-jE\{y\}-jY}{2}$	$\frac{O\{y\}+jE\{y\}+jY}{2}$

4.1.3 O 4-GAC

O GAC 4-usuários admite apenas uma opção de projeto. De fato, todas as autoestruturas devem ser consideradas simultaneamente e a cada usuário é atribuído uma autossequência a partir de um autoespaço distinto. O sinal de saída é dado por

$$y\,[\mathrm{n}] = x_1[\mathrm{n}] + x_2[\mathrm{n}] + x_3[\mathrm{n}] + x_4[\mathrm{n}]\ (mod\ p).$$

Usuários podem ser recuperados a partir de y de acordo com manipulações similares às empregadas para os casos do GAC 2-usuários e 3-usuários. Portanto, após três sucessivas aplicações da TNF para y, o seguinte sistema de equações é obtido:

$$x_1[\mathrm{n}] + x_2[\mathrm{n}] + x_3\,[\mathrm{n}] + x_4[\mathrm{n}]\,(mod\ p) = y[\mathrm{n}], \tag{4.11}$$

$$x_1[\mathrm{n}] - x_2[\mathrm{n}] + jx_3[\mathrm{n}] - jx_4[\mathrm{n}]\,(mod\ p) = \Gamma(y)\,[\mathrm{n}], \tag{4.12}$$

$$x_1[\mathrm{n}] + x_2[\mathrm{n}] - x_3[\mathrm{n}] - x_4[\mathrm{n}]\ (mod\ p) = \Gamma^{(2)}(y)\,[\mathrm{n}], \tag{4.13}$$

$$x_1[\mathrm{n}] - x_2[\mathrm{n}] - jx_3[\mathrm{n}] + jx_4[\mathrm{n}]\ (mod\ p) = \Gamma^{(3)}(y)\,[\mathrm{n}]. \tag{4.14}$$

A solução para o referido sistema é expressa por

$$x_1[\mathrm{n}] = \frac{E\{y\}+E\{Y\}}{2}, \tag{4.15}$$

$$x_2[\mathrm{n}] = \frac{E\{y\}-E\{Y\}}{2}, \tag{4.16}$$

$$x_3[\mathrm{n}] = \frac{O\{y\}-jO\{Y\}}{2}, \tag{4.17}$$

$$x_4[\mathrm{n}] = \frac{O\{y\}+jO\{Y\}}{2}. \tag{4.18}$$

4.2 DISCUSSÃO

Uma vez que se assume que os autovalores usados em um esquema específico são fixos, o sistema de equações a partir do qual as sequências de usuário são recuperadas precisa ser resolvido apenas uma vez. Na verdade, é necessário aplicar a solução cada vez que se recebe uma nova sequência *y*. Porém, tal solução já é conhecida. Além disso, a complexidade computacional deste procedimento pode ser estimada em termos de operações aritméticas. Para isso, deve-se fazer a contagem do número de adições e multiplicações requeridas por "(1)" a solução do sistema de equações e "(2)" o cálculo da transformada de *y* [51].

4.2.1 MÚLTIPLO ACESSO VIA TRANSFORMADAS EM CORPOS FINITOS

Técnicas de acesso múltiplo desempenham um papel essencial nos sistemas de comunicação modernos. O uso simultâneo de um canal por diferentes usuários permite uma arquitetura flexível de tais sistemas e uma alocação razoável dos recursos disponíveis.

Conforme anteriormente observado, o método para separação de sequências apresentado num esquema de comunicação como esse restringe o número de usuários simultâneos pelo número de autovalores distintos que a matriz de transformacão possui. Entretanto, de maneira similar às técnicas de multiplexação usuais, é possível implementar um esquema hierárquico [39]. Isso permite o acesso de um número maior de usuários ao sistema, à medida que novos níveis são criados pela combinação de sinais que se encontram em níveis mais baixos. Portanto, mesmo que uma transformada com um número restrito de autovalores distintos seja usada, a estratégia de criar novos níveis hierárquicos permite multiplexar um número maior de usuários [39]. Estas ideias têm sido estendidas utilizando-se transformadas trigonométricas definidas sobre corpos finitos [17], [20], [39].

Uma primeira estratégia, para se implementar um esquema com 2 níveis, é considerar uma transformada sobre $GF(p_1)$, que é usada no primeiro nível, e outra sobre $GF(p_2)$, que é usada no segundo nível, [17], de forma que a condição $p_2 \geq (p_1)^2$ deve ser respeitada: As combinações $(p_1)^2$ que representam as somas das autossequencias dos usuários dos níveis mais baixos, são mapeadas em autossequências definidas no próximo nível hierárquico. Uma

segunda estratégia seria, ao invés de se realizar um mapeamento extra para um corpo finito primo com característica maior, usar um mapeamento para um corpo de extensão.

A autoestrutura das transformadas trigonométricas também foi utilizada para acesso múltiplo ao RAC. Nesse caso, o aspecto de maior relevância é o fato de que DTTs dos tipos 2 e 3, de comprimento *N*, possuem *N* autovalores distintos (conjectura) [37]. Isso significa que o número de usuários multiplexados não se limita a quatro. Em um trabalho recente, Lima e Campello de Souza usaram como base para um esquema de comunicação multiusuário os autovetores da matriz de transformação da DFrFT como sequências dos usuários [40]. A vantagem de se utilizar a DFrFT é que o número de subespaços gerados, que determina o número máximo de usuários simultâneos de um esquema, é arbitrário. Além disso, as sequências também são construídas a partir de um procedimento sistemático e possuem apenas componentes reais. As DTTs dos tipos 2 e 3, por exemplo, apesar de possuírem números arbitrários de autovalores, possuem autovetores com componentes complexas e cuja obtenção envolve um maior número de cálculos [37].

De acordo com o que foi exposto, teoricamente, é possível o acesso simultâneo ao canal de comunicação de uma quantidade de usuários tão grande quanto se queira. Para isso, basta que se utilize uma transformada a qual esteja associado o número de autovalores necessário para definir o sistema.

Embora a técnica de separação de sequências proposta utilize aritmética modular, em um cenário prático, o ruído que possivelmente corrompe as sequências transmitidas é analógico. Se as sequências estiverem sobre $GF(7)$, por exemplo, deve-se usar sete níveis de quantização, de modo que, mesmo que o canal seja ruidoso, a técnica deve funcionar até um certo nível de relação sinal-ruído (SNR, do inglês, *Signal-to-Noise Ratio*). Sinais podem ser transmitidos utilizando uma modulação PAM (do inglês *pulse amplitude modulation*) 7-ária, cujo desempenho seria basicamente o mesmo de uma PAM 8-ária [55]. Isto indica que o sistema de multiplexação de dados baseado nas transformadas de corpo finito é viável para canais ruidosos. Naturalmente, seu desempenho pode ser melhorado com a utilização de códigos corretores de erro.

Outros aspectos interessantes da comunicação multiusuário baseada nas autoestruturas das transformadas de corpo finito podem ser revelados através de uma comparação com o múltiplo acesso por divisão de códigos usando o espalhamento espectral direto sobre sequências [41].

Nesse sentido, os esquemas apresentados podem ser vistos como DS-CDMA em que as sequências de espalhamento (assinaturas dos usuários) são autovetores de uma transformada sobre corpo finito, em vez de códigos de Walsh ou sequências pseudo-aleatórias.

Diferentemente dos receptores DS-CDMA, que usam propriedades de autocorrelação e de correlação cruzada das sequências de espalhamento, na proposta apresentada neste trabalho, o sucesso da separação de cada sequência de usuário depende da ortogonalidade entre autoespaços distintos.

Além disso, como o DS-CDMA utiliza sequências binárias, ele aumenta simultaneamente a taxa de transmissão e a largura de banda pelo mesmo fator, mantendo, assim, a eficiência espectral inalterada. Esta abordagem explora propriedades de ortogonalidade de sequências não-binárias (multinível) sobre corpos finitos. A principal vantagem desta técnica em relação a esquemas clássicos de multiplex ou acesso múltiplo digital é sua melhor eficiência espectral, em particular, para canais com alta SNR [8].

CAPÍTULO 5
SIMULAÇÕES E RESULTADOS

O objetivo deste capítulo é apresentar uma avaliação do desempenho do sistema de múltiplo acesso 2-GAC, sob a influência do ruído aditivo Gaussiano branco, por meio de simulação computacional. Para isso, a simulação do sistema em dois cenários distintos, para fins de comparação do desempenho, é implementada por meio do aplicativo Simulink/MatLab™. A simulação do sistema é feita em duas situações distintas. Na primeira, o canal $GF(p)$ aditivo com dois usuários é avaliado sem o uso de codificação de canal, e na segunda situação, adiciona-se a codificação de canal via códigos de Fourier que utilizam os algoritmos de decodificação apresentados no Capítulo 3. O desempenho do sistema é avaliado em termos de taxa de erro de símbolo *versus* relação sinal-ruído e comparado entre os dois cenários simulados.

5.1 O SIMULADOR

O simulador é composto por rotinas desenvolvidas na linguagem *Matlab* e de blocos funcionais criados com a ferramenta *Simulink*. Na primeira etapa do trabalho, são realizadas simulações para verificação da correta operação do modelo implementado, visando comparar os resultados da metodologia utilizada com os exemplos teóricos, disponíveis em [32], [38], [44]. O sistema é composto pelos módulos apresentados nas seções seguintes.

5.2 O DIAGRAMA DO MÓDULO DE TRANSMISSÃO

A Figura 5.1 ilustra os blocos funcionais que compõem o módulo de transmissão, tendo como entradas os dados dos usuários. O transmissor recebe as sequências de símbolos aleatórios provenientes da fonte de informação ASCII representando os dados dos usuários. Esses dados, então, são convertidos (conversão caracter ASCII → decimal) em decimais que posteriormente são mapeados em autossequências as quais funcionam como assinaturas de usuários (autossequência associada a autovalor distinto). As autossequências são então

adicionadas no 2-GAC e a sequência obtida, $y = x_1 + x_2$ (*mod p*), é transmitida como indicado na Figura 5.1. O número total de símbolos transmitidos é calculado em função da quantidade de iterações. As funções dos blocos do módulo de transmissão são habilitadas por meio de rotinas específicas para execução destas funções. A fonte de informação utiliza o código ASCII para transmitir a informação desejada. O codificador ASCII executa a função do codificador de fonte, tendo como entradas a sequência de informação de cada usuário e como saídas o valor em decimal que representa o caracter ASCII.

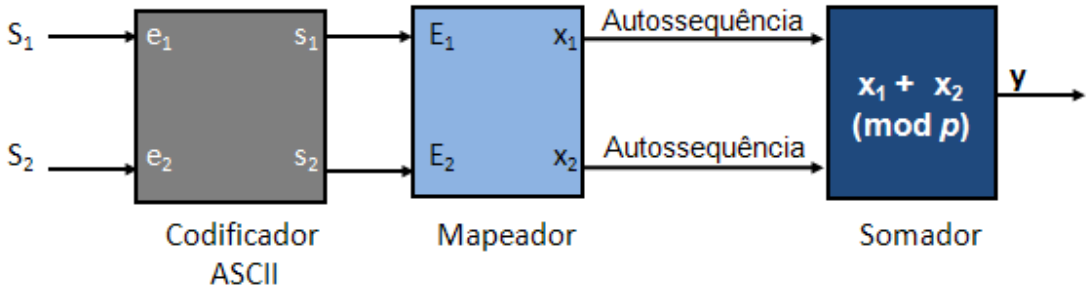

Figura 5.1 - *Diagrama em blocos do módulo de transmissão*

5.2.1 PROCESSO DE CODIFICAÇÃO DE CANAL

O propósito da codificação de canal é introduzir na sequência de informação uma determinada quantidade de informação redundante, de forma que, no receptor, esta informação redundante possa ser utilizada para detectar e corrigir erros decorrentes de ruído e interferência que afetam o sinal quando este é transmitido através do canal de transmissão.

No canal *GF*(*p*) aditivo de dois usuários da simulação, a codificação de canal é realizada pelos códigos de Fourier [43], [44], descritos no Capítulo 3, que são códigos de bloco lineares que operam com símbolos não-binários. Os dados de entrada são agrupados em k símbolos, com $k = 2$, formando a mensagem. Cada mensagem de k símbolos é multiplicada pela matriz geradora do código formando uma sequência de comprimento $n = 7$. Esta sequência é denominada de palavra-código, tal que cada mensagem seja univocamente relacionada com a respectiva palavra-código. As palavras-código assim geradas são autossequências da TNF e funcionam como assinaturas de usuários.

Neste trabalho, os códigos de Fourier usados são o código $F^1(7,2,5)$ para o usuário 1 e $F^{-1}(7,2,5)$ para o usuário 2, ambos definidos sobre $GF(29)$. A Figura 5.2 ilustra uma representação simplificada da codificação Fourier, resultando em um sinal codificado de *7* (sete) símbolos com capacidade de corrigir até dois símbolos recebidos incorretamente. Nesta simulação, é implementado um bloco funcional responsável pela codificação da mensagem dos usuários nas palavras-código, predefinidas, apresentadas no Anexo A.

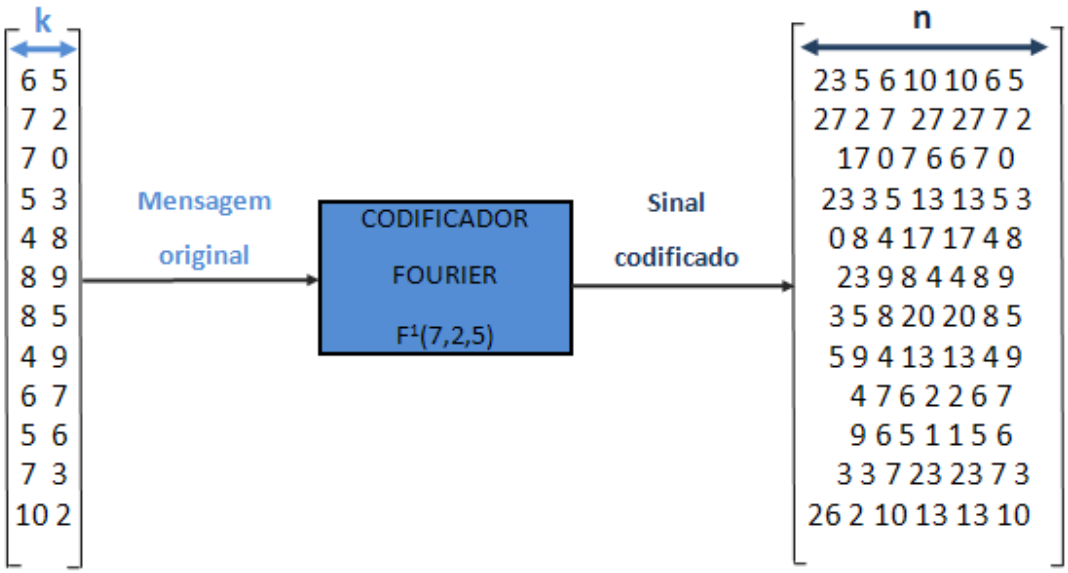

Figura 5.2 - *Representação da codificação Fourier*.

5.2.2 O GERADOR DE SÍMBOLOS ALEATÓRIOS

O gerador de símbolos aleatórios é implementado pela rotina "*fonte_ aleatória*" por meio do comando *rand* do *Matlab*, sendo a primeira tarefa executada pelo simulador e que tem como função a geração de símbolos uniformemente distribuídos entre os caracteres ASCII imprimíveis mostrados na Tabela B.1 no Apêndice B. A rotina gera 32 caracteres para cada usuário, por iteração.

5.2.3 O CODIFICADOR ASCII

O codificador ASCII é implementado pela rotina *"codificador_ASCII"*. Esta rotina é utilizada para identificar o caracter ASCII gerado pela rotina "*fonte_ aleatória*" e codificá-lo em sua representação decimal, de acordo com a Tabela B.1 mostrada no Apêndice B.

5.2.4 O CODIFICADOR FOURIER

No codificador Fourier, apresentado na Figura 5.3, os símbolos provenientes do codificador ASCII são relacionados por meio de uma tabela a uma palavra-código. O bloco é implementado utilizando o bloco *Embedded MATLAB Function*, da biblioteca *user-defined functions* do *Simulink*. Como entrada tem-se a representação em decimal dos correspondentes caracteres ASCII, provenientes do codificador ASCII, e, como saídas, as respectivas palavras-código de cada usuário, de dimensões [1 x 7] por caracter.

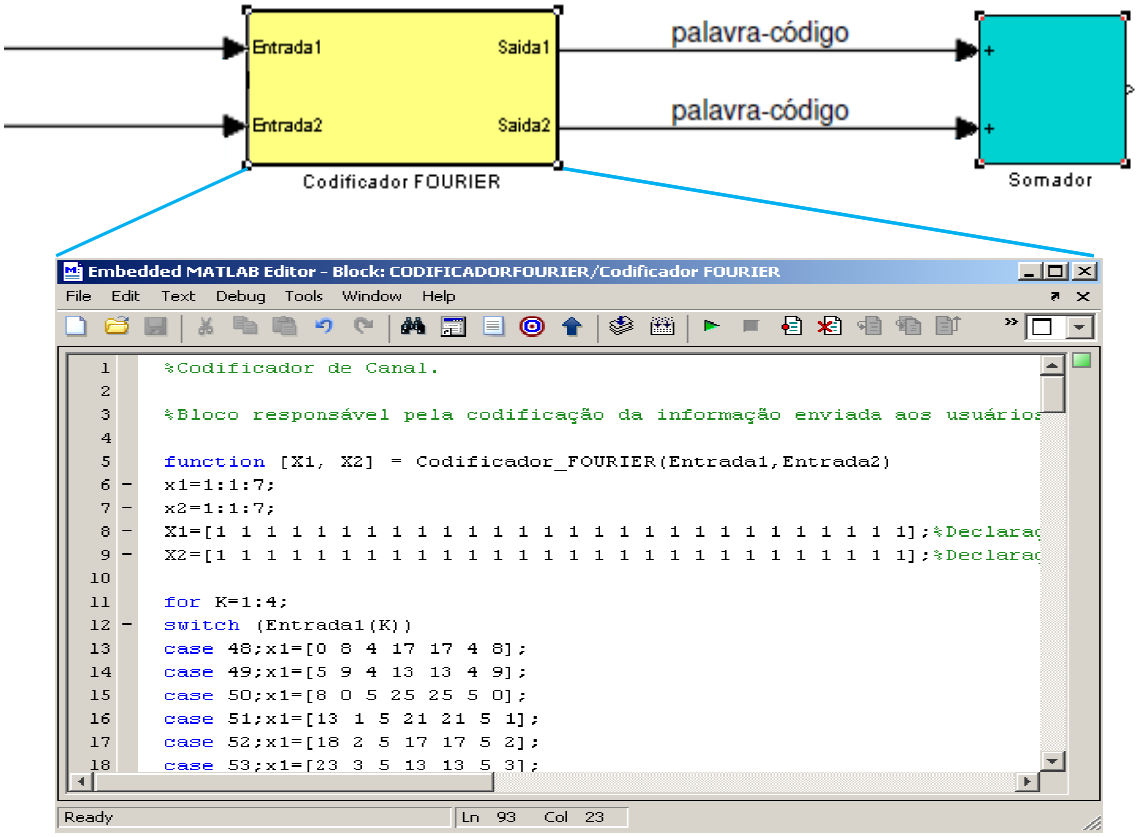

Figura 5.3 – *O Codificador Fourier*

5.3 O CANAL RGBA

O canal RGBA mostrado na Figura 5.4, representa o canal de ruído Gaussiano branco aditivo ou canal RGBA. Sua função é adicionar ruído ao sinal de entrada, simulando assim um sistema real cuja informação sofre degradação quando submetida ao meio de transmissão. Este bloco possui como entrada a sequência $y = x_1 + x_2$ (*mod p*), e, como sinal de saída do bloco, tem-se a sequência *r*, em que $r = y + e$ (mod *p*) é o sinal transmitido com adição de ruído, o qual será enviado à entrada do módulo de recepção.

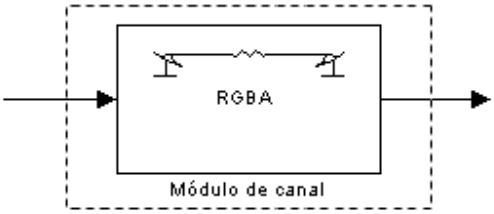

Figura 5.4 - *Diagrama em bloco do módulo de canal*

5.4 O DIAGRAMA DO MÓDULO DE RECEPÇÃO

Esse bloco possui como entradas as autossequências com adição de ruído, e como saídas os possíveis valores em decimal dos caracteres ASCII transmitidos. Aqui, discutem-se as ações realizadas pelo receptor, que coleta o sinal após a passagem pelo canal RGBA e executa as operações necessárias para identificação das autossequências associadas a cada usuário e à recuperação das informações transmitidas.

Conforme mostra a Figura 5.5, este módulo é composto pelos blocos funcionais FFT, somadores, multiplicadores, mapeador e decodificador ASCII. A primeira ação consiste em obter o espectro da sequência recebida, sequência $r = y + e$ (mod *p*). Para isto utilizam-se algoritmos FFT, em que se obtém $\boldsymbol{R} = \boldsymbol{X}_1 + \boldsymbol{X}_2$ (mod *p*) (considerando $e = 0$) em que $\boldsymbol{X}_1 = \lambda_1 x_1 = x_1$ e $\boldsymbol{X}_2 = \lambda_2 x_2 = -x_2$, ou seja, $\boldsymbol{R} = x_1 - x_2$. No que se segue, nos somadores obtém-se as sequências $r + \boldsymbol{R}$ (mod *p*) $= 2x_1$ e $r - \boldsymbol{R}$(mod *p*) $= 2x_2$, e após as multiplicações as autossequências dos usuários 1 e 2 são recuperadas: $x_1 = r - \boldsymbol{R}$(mod *p*) × 15 e

$x_2 = r - \boldsymbol{R}(\text{mod } p) \times 15$, em que $15 \equiv 2^{-1}(\text{mod } 29)$. As mesmas tabelas empregadas na transmissão são utilizadas na recepção, conforme valores indicados na Tabela A.1 e na Tabela A.2, respectivamente, para os usuários 1 e 2, apresentadas no Anexo A.

No receptor, uma vez que as autossequências de cada usuário são separadas, (a tabela construída permite identificar o caracter ASCII que corresponde à autossequência recebida), as informações sobre a mensagem transmitida e a identificação do usuário associada com ela são recuperadas. Aqui, as sequências de entrada inicialmente passam por arredondamentos, pois os algoritmos apenas permitem identificar sequências de elementos inteiros, e posteriormente o tipo de decisão selecionado é decisão abrupta, em que, se o elemento do vetor for negativo, atribui-se o "0" e, se o elemento do vetor for igual a 29, atribui-se o "28". Desta forma, limita-se os elementos da sequência recebida aos números inteiros não negativos, com valores de 0 a p - 1.

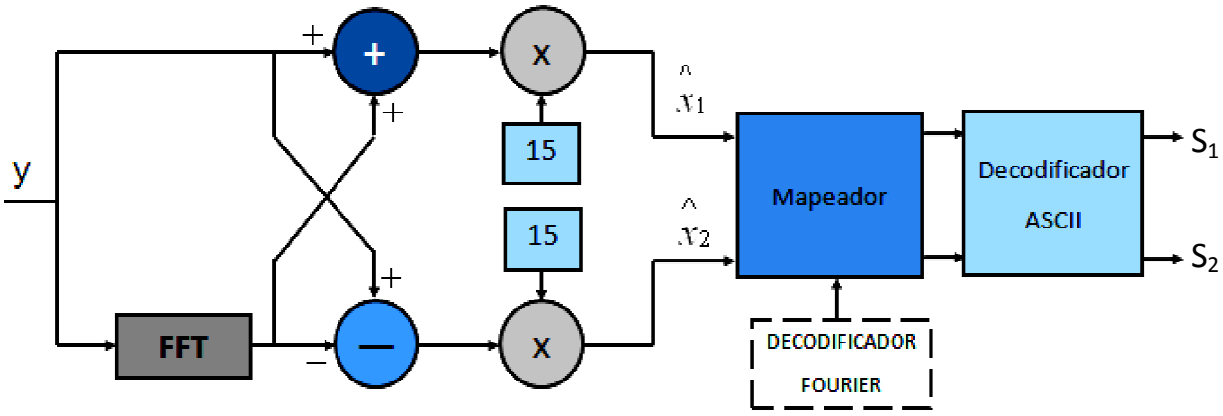

Figura 5.5 - *Diagrama em blocos do módulo de recepção.*

5.4.1 PROCESSO DE DECODIFICAÇÃO DE CANAL

No segundo cenário simulado é adicionada a codificação de canal. Então, o estágio final do módulo de recepção é a realização da decodificação de canal (o vetor das autossequências dos usuários é enviado à entrada do decodificador Fourier). O decodificador analisa as autossequências provenientes do canal de propagação e, segundo o método de codificação utilizado na transmissão, tenta reproduzir os símbolos originalmente gerados. O decodificador é

composto pelos processos de detecção e correção de erros, ilustrados no diagrama em blocos da Figura 5.6.

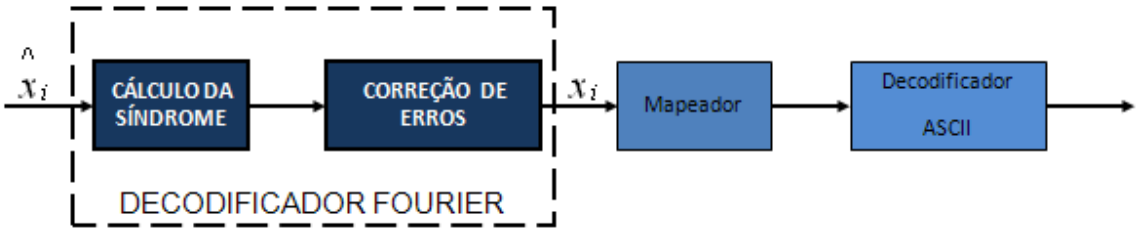

Figura 5.6 - *Diagrama em blocos do processo de decodificação.*

Assim como no transmissor, estes processos são habilitados nos blocos funcionais individualmente de acordo com o programa da simulação. Desta forma, o decodificador faz a decodificação de um sinal de comprimento *n* em uma mensagem de comprimento *k*. A Figura 5.7, mostra uma representação simplificada da decodificação executada para o caso $n = 7$, $k = 2$ e $d = 5$.

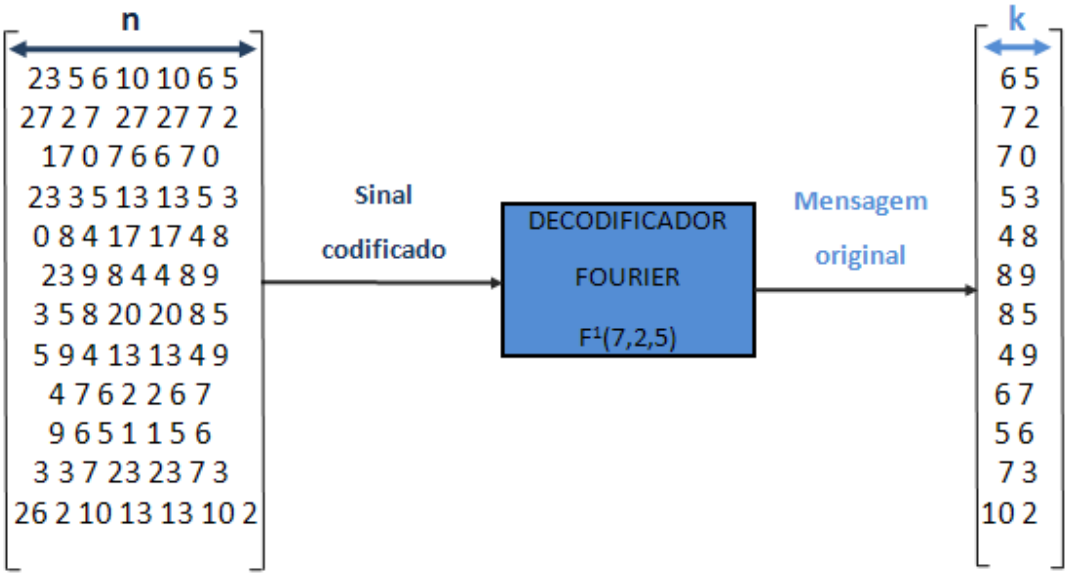

Figura 5.7 - *Representação da decodificação Fourier*.

5.4.2 O DECODIFICADOR FOURIER

Esse bloco, o decodificador Fourier apresentado na Figura 5.8, possui como entradas as autossequências x_1 e x_2 respectivamente, recebidas contendo eventuais erros, após a passagem pelo canal RGBA. Desde que a sequência recebida é uma autossequência presente na tabela, seu respectivo caracter ASCII é imediatamente identificado, caso contrário, os algoritmos de decodificação, que visam à correção de erros, são aplicados, a fim de permitir a correta decodificação dos símbolos recebidos. Em particular, implementou-se os algoritmos de decodificação proposto por Freire et al [43], [44], para correção de um e dois erros no 2-GAC.

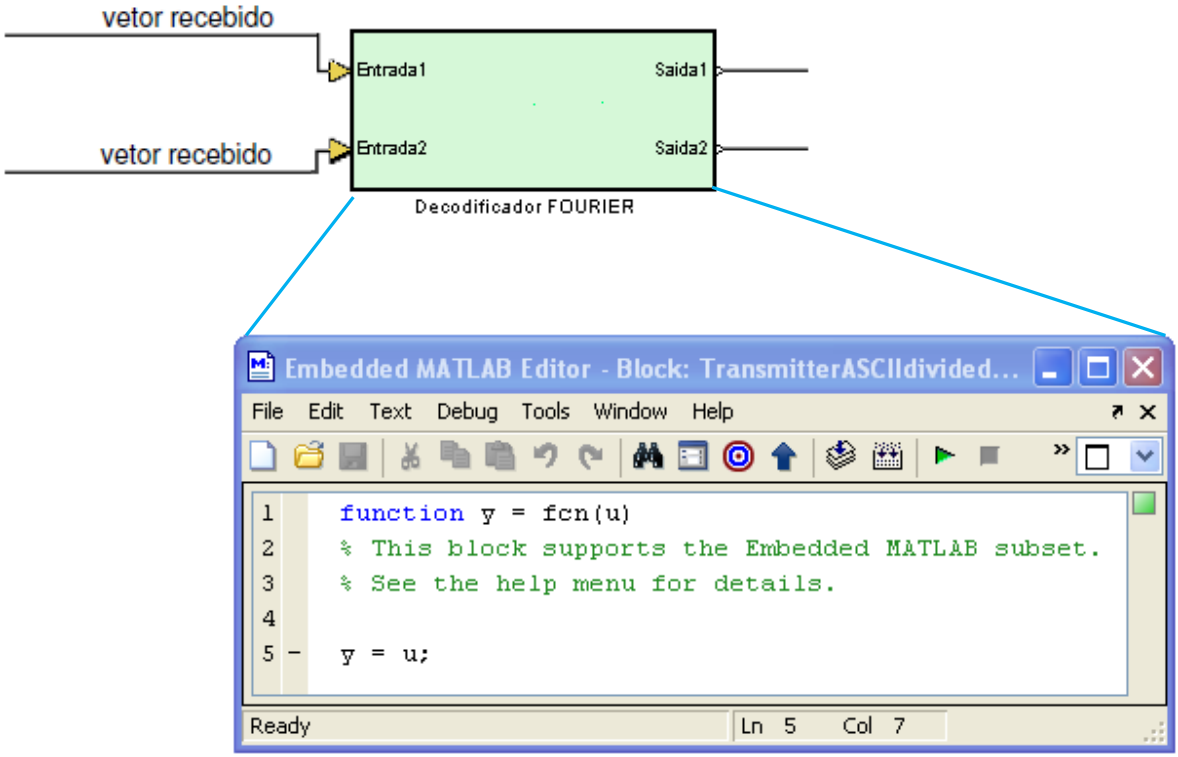

Figura 5.8 - *Decodificador Fourier*.

5.4.3 PROCESSO DE DETECÇÃO DE ERROS

O processo de detecção de erros analisa as autossequências recebidas através do cálculo da síndrome, permitindo assim definir se estas autossequências fazem parte da lista do código do respectivo usuário, ou se devem sofrer correções provenientes de erros induzidos pelo canal durante a transmissão.

Considere a sequência recebida $r = x_i + e \pmod{p}$, em que $x_1 \in F^1(7, 2, 5)$, $x_2 \in F^{-1}(7, 2, 5)$ e e denota a sequência erro, com elementos sobre $GF(29)$. A síndrome de r é definida como $S \triangleq Fr - \lambda r$, que é zero se, e somente se, r é uma autossequência da TNF associada ao autovalor λ. Uma transformada rápida é utilizada para calcular S e verificar se a sequência recebida pertence ao código ou não. A Figura 5.9 ilustra o bloco funcional que compõe o processo de detecção de erros.

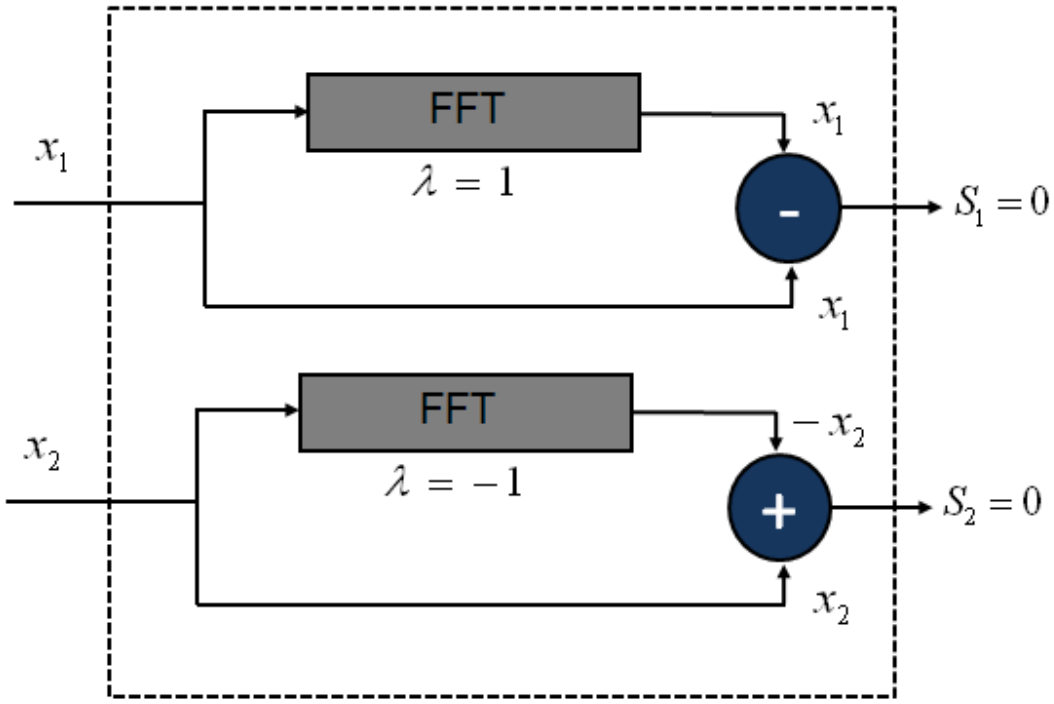

Figura 5.9 - *Diagrama em bloco do módulo de detecção de erros.*

5.4.4 PROCESSO DE CORREÇÃO DE ERROS

Aqui são traduzidas em rotinas de programação todas as etapas dos algoritmos de decodificação para correção de um único erro e de dois erros, baseados na autoestrutura da TNF, conforme descrito no Capítulo 3, de modo que o receptor possa utilizar esta característica na detecção e/ou correção de erros.

5.5 MÉTRICAS DE DESEMPENHO DO SISTEMA

Para a avaliação do desempenho do sistema 2-GAC com codificação de canal, são utilizadas essencialmente métricas associadas à capacidade de correção de erros do código de Fourier. Os dados recebidos são comparados com aqueles gerados originalmente no transmissor. Todos os erros são contados e a taxa de erro de símbolo é calculada como a razão entre símbolos errados e o total de símbolos enviados. Essa técnica é conhecida como método de Monte Carlo [52]. A taxa de erro observada deve estar associada a um grau de confiabilidade, ou seja, devem ser determinadas estatísticas com intervalos de confiança.

Se *NS* é o número de símbolos transmitidos e *n* é o número correspondente de erros observados, então, uma estimativa imparcial da taxa de erro da simulação é $\hat{p} = n/NS$ [53]. Quando o número de símbolos tende para a condição limítrofe, ou seja, quando $NS \rightarrow \infty$, a estimativa $\hat{p}$ irá convergir para *p*, em que *p* é a probabilidade de erro de símbolo verdadeira. Com a finalidade de fornecer uma visão concisa sobre a resposta do sistema frente às condições impostas pelo canal RGBA, realizou-se uma avaliação da taxa de erro de símbolo *versus* SNR dos dados obtidos no receptor.

Assim, é preciso definir o valor de *NS* de forma adequada, a fim de obter uma aproximação razoável para a taxa de erro. Para valores finitos de *NS*, pode-se quantificar a confiabilidade da estimativa em termos dos intervalos de confiança. De acordo com [52], esse intervalo de confiança (y_-, y_+) é dado por

$$y_{\pm} = 10^{-k}\{1+(d^2_{\alpha}/2\eta)[1\pm(4\eta/d^2_{\alpha}+1)^{1/2}]\} \tag{5.1}$$

em que = 10^{-k} e $NS = \eta 10^{k}$ e d_α é escolhido de forma a satisfazer os níveis de confiança de 90% ($d_\alpha = 1{,}64$), 95% ($d_\alpha = 1{,}96$) e 99% ($d_\alpha = 2{,}58$). Este intervalo é exibido na Figura 5.10 [52] para os intervalos de confiança de 90%, 95% e 99%.

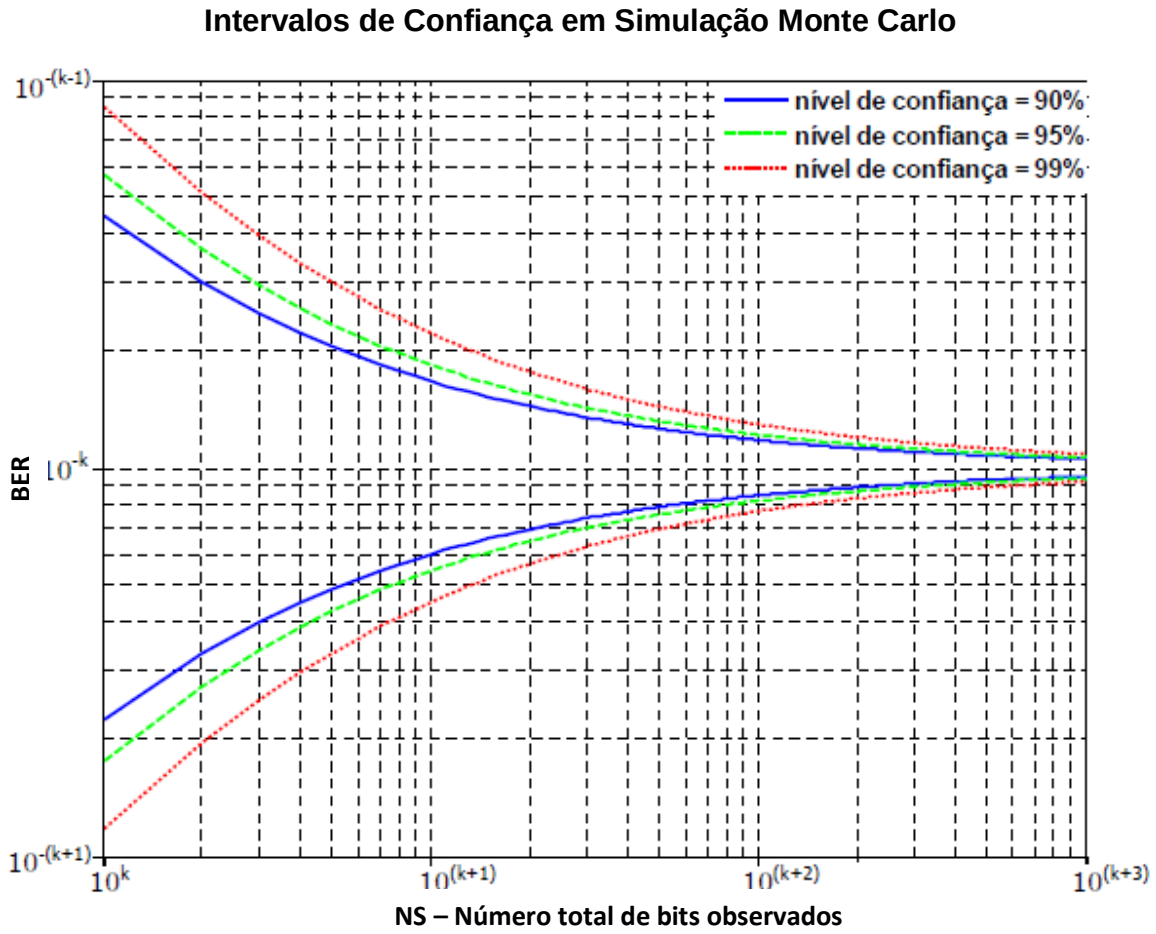

Figura 5.10 – *Intervalos de confiança de taxa de erro quando o valor observado for 10^{-k}, para simulações que utilizam o método de Monte Carlo.*

Observando essas curvas, constata-se que os intervalos de confiança diminuem lentamente com o crescimento do número de erros *n*. Para a simulação descrita nesta dissertação, foi adotado o nível de confiança de 99% e um número de símbolos processados igual a $6{,}4 \times 10^{6}$. Dessa forma, a estimativa de taxa de erro de símbolo ficará limitada ao intervalo aproximado [2,7 ; 0,39].

5.6 CENÁRIOS DE AVALIAÇÃO

Dois cenários diferentes de avaliação são elaborados. Esses cenários de testes são construídos por meio da variação das características do simulador. Na simulação, utilizou-se uma função que permite especificar a SNR ao qual a transmissão está submetida. Em todos os casos, o procedimento para simular o canal Gaussiano é repetido para valores de SNR na faixa de 2dB a 14dB.

Em uma primeira etapa, a configuração inicial é composta por um canal sob efeito do ruído, sem a presença do código corretor de erros no canal de transmissão, em que a eficiência do sistema depende apenas da relação sinal-ruído do canal.

Para cada valor de SNR considerado, é obtida a razão entre o número total de símbolos recebidos com erro e o número total de símbolos gerados na simulação. Os resultados obtidos, que confirmam o comportamento previsto na discussão realizada, são apresentados na Figura 5.11.

O outro cenário configurado adiciona o efeito do esquema de codificação de canal via códigos de Fourier. Neste cenário, é explorado o funcionamento do esquema de detecção e correção de erros do código; nesse caso, o desempenho do sistema depende da relação sinal-ruído do canal e dos esquemas de decodificação do código corretor de erro.

5.7 AVALIAÇÃO DOS EFEITOS DO ESQUEMA DE CODIFICAÇÃO DE CANAL VIA CÓDIGOS DE FOURIER.

Nesta seção efetua-se a comparação entre os resultados conseguidos com a utilização do esquema de decodificação e correção de erros do código de Fourier, frente a um esquema sem utilização do código corretor. Os resultados são expressos através de curvas de probabilidade de erro de símbolo *versus* relação sinal-ruído (SNR) do canal. As curvas são validadas calculando-se os intervalos de confianca a 99% para os pontos obtidos na simulação [52]. As convergências são garantidas pela versão forte da lei dos grandes números. Para tanto, são avaliadas as curvas de taxa de erro de símbolo *versus* SNR, comparando a resposta dos modelos simulados em canais submetidos ao RGBA.

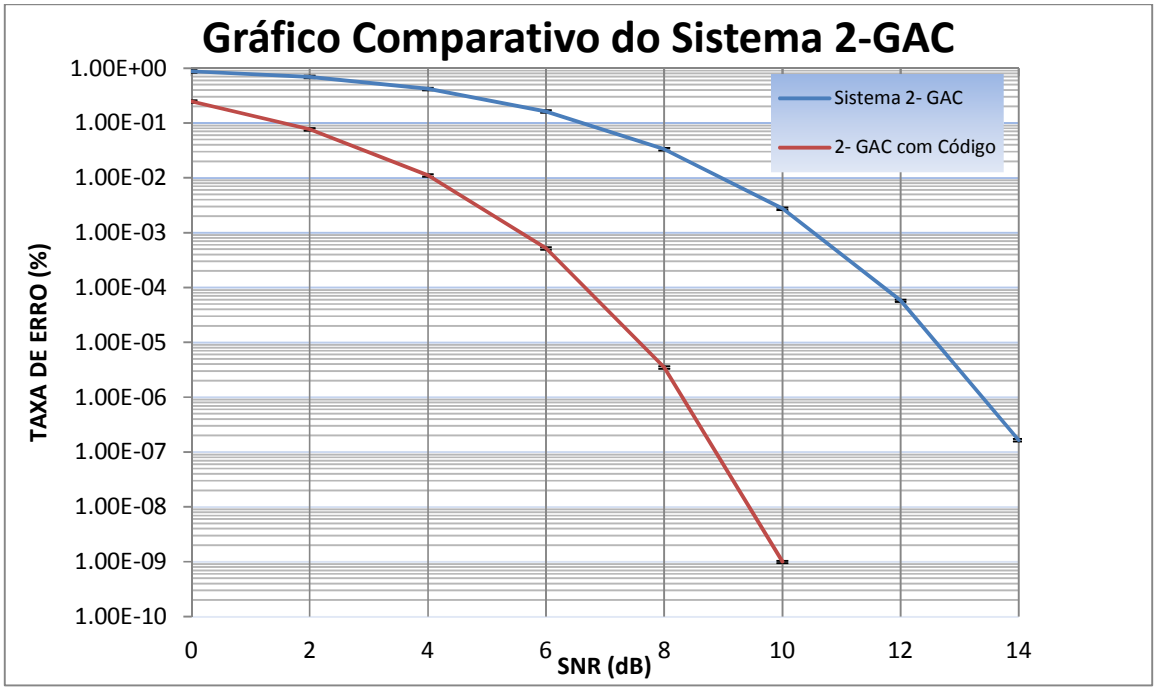

Figura 5.11- *Curvas SNR versus taxa de erro de símbolo do 2-GAC.*

Considera-se nesta simulação um esquema com 2 usuários baseado na TNF de comprimento $N = 7$. Com o auxilio do Matlab, as autossequências, associadas aos autovalores $\lambda = 1$ e $\lambda = -1$, para os usuários 1 e 2, respectivamente, são obtidas. No 2-GAC com codificação de canal, utiliza-se a TNF de comprimento $n = 7$, sobre $GF(29)$. Usando-se os códigos, $F^1(7,2)$ para o usuário1 e $F^{-1}(7,2)$ para o usuário 2, respectivamente, os esquemas de decodificação usados são baseados na autoestrutura da TNF. Nesse cenário, as condições de simetria e a possibilidade de usar algoritmos rápidos para calcular a TNF desempenham um papel importante.

CAPÍTULO 6
CONCLUSÕES

6.1 APLICAÇÕES DA AUTOESTRUTURA DA TRANSFORMADA NUMÉRICA DE FOURIER

Essa dissertação considera a autoestrutura da transformada de Fourier definida sobre $GF(p)$, a qual é denominada transformada numérica de Fourier (TNF). Aplicações das autossequências do operador TNF unitário nas áreas de Codificação de Canal [42] e Comunicação Multiusuário [38] são consideradas, em que as mesmas são usadas, respectivamente, como palavras de um código de bloco linear p-ário e como assinaturas de usuário para acesso múltiplo ao canal $GF(p)$ aditivo. Nesse cenário, um sistema de múltiplo acesso proposto recentemente na literatura foi analisado e implementado por meio de simulação computacional. Para efeito de comparação, o desempenho do sistema em termos de taxa de erro de símbolo versus relação sinal-ruído foi avaliado em dois cenários distintos, a saber, com e sem o uso de codificação de canal.

6.2 CONTRIBUIÇÕES

Embora o objetivo deste trabalho não é uma descrição completa de um cenário prático de comunicação multiusuário, é relevante mencionar um dos seus requisitos, uma vez que, para a maioria dos canais práticos não se pode assumir ausência de ruído e uma análise de erro é necessária. A análise de desempenho do sistema em termos da taxa de erro de símbolo é realizada considerando que o canal de transmissão está sob o efeito do ruído Gaussiano branco aditivo. Os procedimentos de decodificação para correção de um e dois erros baseados na autoestrutura da TNF são utilizados para avaliar o sistema com codificação de canal, em que os algoritmos realizam apenas cálculos da TNF (realizados através da transformada rápida de Fourier) e algumas operações simples tais como, adições, substituições e multiplicações por constantes. Mostrou-se, por meio de simulações, que o uso das autossequências da TNF para canais submetidos ao RGBA é viável, haja vista que as sequências de cada usuário são

recuperadas mesmo sem a utilização da codificação de canal. A simulação é a metodologia utilizada, a fim de investigar o desempenho do sistema 2-GAC. O sistema é implementado e avaliado por simulação computacional em dois modos de operação diferente. O desempenho apresentado pelo sistema 2-GAC, sob o efeito do RAGB, sem o uso dos códigos de Fourier é comparado ao desempenho do 2-GAC com o uso de codificação de canal. O método de avaliação é concentrado na análise das curvas em termos de taxa de erro de símbolo *versus* relação sinal-ruído, utilizadas para comparar o desempenho nos dois diferentes modos de operação. Comparando-se a resposta do modelo em um canal de transmissão sob os efeitos da codificação de canal via códigos de Fourier com o comportamento do sistema simulado em um canal RGBA sem controle de erros, é possível perceber a influência dos efeitos do código nos valores da taxa de erro de símbolo, para todos os valores de SNR do canal.

Tomando como referência, em um canal RGBA, uma SNR de 8dB, observa-se uma taxa de erro de símbolo de 10^{-6} com o emprego do código. Contudo, para o canal sem o esquema de codificação a relação sinal-ruído que garanta tal taxa de erro de símbolo é a partir de uma SNR de 13dB. Conforme esperado, analisando-se os resultados obtidos, que estão ilustrados nas curvas da Figura 5.11, verifica-se que o desempenho do sistema sem codificação de canal é menos eficiente do que com esta.

6.3 SUGESTÕES PARA TRABALHOS FUTUROS

Para dar continuidade às investigações realizadas nesse trabalho, são sugeridas as linhas de ação descritas a seguir.

i) Concepção de estratégias de correção de três erros para os códigos de Fourier, baseadas nas características de simetria das autossequências da TNF.

ii) Concepção de estratégias de correção de números arbitrários de erros para os códigos de Fourier, baseadas na autoestrutura da TNF.

iii) Análise da utilização, no sistema multiusuário considerado nesse trabalho, de transformadas numéricas de Fourier sem multiplicações, visando reduzir a complexidade computacional de sua implementação.

iv) Implementação, por meio de simulação computacional, e avaliação de desempenho dos sistemas multiusuário sobre o canal GF(p) aditivo com três e quatro usuários.

v) Investigação de esquemas hierárquicos para número arbitrário de usuários.

vi) Análise e implementação, por meio de simulação computacional, de sistemas multiusuários semelhantes ao descrito nessa dissertação, porém baseados na autoestrutura de outras transformadas de corpo finito [40], [54], tais como as transformadas do cosseno e do seno de corpo finito e a transformada de Fourier fracionária de corpo finito [54].

APÊNDICE A

TABELAS COM AS PALAVRAS-CÓDIGO DOS USUÁRIOS

Este apêndice mostra as tabelas com as palavras-código, usadas para o usuário 1 e usuário 2, dos códigos apresentados no Capítulo 4.

Tabela A.1 - *Código de Fourier* $F^{1}(7, 2, 5)$ *do usuário 1,* $\alpha=7$*, sobre* $GF(29)$*, com* $\sqrt{7}(mod\ 29) = 23$

Palavra-código	*Mensagem*
9 2 3 7 7 3 2	3 2
14 3 3 3 3 3 3	3 3
19 4 3 28 28 3 4	3 4
24 5 3 24 24 3 5	3 5
0 6 3 20 20 3 6	3 6
5 7 3 16 16 3 7	3 7
10 8 3 12 12 3 8	3 8
15 9 3 8 8 3 9	3 9
18 0 4 20 20 4 0	4 0
23 1 4 16 16 4 1	4 1
28 2 4 12 12 4 2	4 2
4 3 4 8 8 4 3	4 3
9 4 4 4 4 4 4	4 4
14 5 4 0 0 4 5	4 5
19 6 4 25 25 4 6	4 6
24 7 4 21 21 4 7	4 7
0 8 4 17 17 4 8	4 8
5 9 4 13 13 4 9	4 9
8 0 5 25 25 5 0	5 0
13 1 5 21 21 5 1	5 1
18 2 5 17 17 5 2	5 2
23 3 5 13 13 5 3	5 3
28 4 5 9 9 5 4	5 4
4 5 5 5 5 5 5	5 5
9 6 5 1 1 5 6	5 6
14 7 5 26 26 5 7	5 7
19 8 5 22 22 5 8	5 8
24 9 5 18 18 5 9	5 9
27 0 6 1 1 6 0	6 0
3 1 6 26 26 6 1	6 1
8 2 6 22 22 6 2	6 2
13 3 6 18 18 6 3	6 3
18 4 6 14 14 6 4	6 4
23 5 6 10 10 6 5	6 5
28 6 6 6 6 6 6	6 6
4 7 6 2 2 6 7	6 7
9 8 6 27 27 6 8	6 8
14 9 6 23 23 6 9	6 9
17 0 7 6 6 7 0	7 0
22 1 7 2 2 7 1	7 1
27 2 7 27 27 7	7 2
3 3 7 23 23 7 3	7 3
8 4 7 19 19 7 4	7 4
13 5 7 15 15 7 5	7 5

Palavra-código	*Mensagem*
18 6 7 11 11 7 6	7 6
23 7 7 7 7 7 7	7 7
28 8 7 3 3 7 8	7 8
4 9 7 28 28 7 9	7 9
7 0 8 11 11 8 0	8 0
12 1 8 7 7 8 1	8 1
17 2 8 3 3 8 2	8 2
22 3 8 28 28 8 3	8 3
27 4 8 24 24 8 4	8 4
3 5 8 20 20 8 5	8 5
8 6 8 16 16 8 6	8 6
13 7 8 12 12 8 7	8 7
18 8 8 8 8 8 8	8 8
23 9 8 4 4 8 9	8 9
26 0 9 16 16 9 0	9 0
2 1 9 12 12 9 1	9 1
7 2 9 8 8 9 2	9 2
12 3 9 4 4 9 3	9 3
17 4 9 0 0 9 4	9 4
22 5 9 25 25 9 5	9 5
27 6 9 21 21 9 6	9 6
3 7 9 17 17 9 7	9 7
8 8 9 13 13 9 8	9 8
13 9 9 9 9 9 9	9 9
16 0 10 21 21 10 0	10 0
21 1 10 17 17 10 1	10 1
26 2 10 13 13 10 2	10 2
2 3 10 9 9 10 3	10 3
7 4 10 5 5 10 4	10 4
12 5 10 1 1 10 5	10 5
17 6 10 26 26 10 6	10 6
22 7 10 22 22 10 7	10 7
27 8 10 18 18 10 8	10 8
3 9 10 14 14 10 9	10 9
6 0 11 26 26 11 0	11 0
11 1 11 22 22 11 1	11 1
16 2 11 18 18 11 2	11 2
21 3 11 14 14 11 3	11 3
26 4 11 10 10 11 4	11 4
2 5 11 6 6 11 5	11 5
7 6 11 2 2 11 6	11 6
12 7 11 27 27 11 7	11 7
17 8 11 23 23 11 8	11 8
22 9 11 19 19 11 9	11 9
25 0 12 2 2 12 0	12 0
1 1 12 27 27 12 1	12 1
6 2 12 23 23 12 2	12 2
11 3 12 19 19 12 3	12 3
16 4 12 15 15 12 4	12 4
21 5 12 11 11 12 5	12 5
26 6 12 7 7 12 6	12 6

Tabela A.2 - *Código de Fourier $F^{-1}(7, 2, 5)$ do usuário 2, $\alpha=7$, sobre GF(29), com $\sqrt{7}(mod\ 29)$ = 23.*

Palavra-código	*Mensagem*
1 2 3 12 12 3 2	3 2
21 3 3 3 3 3 3	3 3
12 4 3 23 23 3 4	3 4
3 5 3 14 14 3 5	3 5
23 6 3 5 5 3 6	3 6
14 7 3 25 25 3 7	3 7
5 8 3 16 16 3 8	3 8
25 9 3 7 7 3 9	3 9
6 0 4 11 11 4 0	4 0
26 1 4 2 2 4 1	4 1
17 2 4 22 22 4 2	4 2
8 3 4 13 13 4 3	4 3
28 4 4 4 4 4 4	4 4
19 5 4 24 24 4 5	4 5
10 6 4 15 15 4 6	4 6
1 7 4 6 6 4 7	4 7
21 8 4 26 26 4 8	4 8
12 9 4 17 17 4 9	4 9
22 0 5 21 21 5 0	5 0
13 1 5 12 12 5 1	5 1
4 2 5 3 3 5 2	5 2
24 3 5 23 23 5 3	5 3
15 4 5 14 14 5 4	5 4
6 5 5 5 5 5 5	5 5
26 6 5 25 25 5 6	5 6
17 7 5 16 16 5 7	5 7
8 8 5 7 7 5 8	5 8
28 9 5 27 27 5 9	5 9
9 0 6 2 2 6 0	6 0
0 1 6 22 22 6 1	6 1
20 2 6 13 13 6 2	6 2
11 3 6 4 4 6 3	6 3
2 4 6 24 24 6 4	6 4
22 5 6 15 15 6 5	6 5
13 6 6 6 6 6 6	6 6
4 7 6 26 26 6 7	6 7
24 8 6 17 17 6 8	6 8
15 9 6 8 8 6 9	6 9
25 0 7 12 12 7 0	7 0
16 1 7 3 3 7 1	7 1
7 2 7 23 23 7 2	7 2
27 3 7 14 14 7 3	7 3
18 4 7 5 5 7 4	7 4
9 5 7 25 25 7 5	7 5
0 6 7 16 16 7 6	7 6
20 7 7 7 7 7 7	7 7
11 8 7 27 27 7 8	7 8
2 9 7 18 18 7 9	7 9
12 0 8 22 22 8 0	8 0
3 1 8 13 13 8 1	8 1
23 2 8 4 4 8 2	8 2

Palavra-código	*Mensagem*
14 3 8 24 24 8 3	8 3
5 4 8 15 15 8 4	8 4
25 5 8 6 6 8 5	8 5
16 6 8 26 26 8 6	8 6
7 7 8 17 17 8 7	8 7
2 7 8 8 8 8 8 8	8 8
18 9 8 28 28 8 9	8 9
28 0 9 3 3 9 0	9 0
19 1 9 23 23 9 1	9 1
10 2 9 14 14 9 2	9 2
1 3 9 5 5 9 3	9 3
21 4 9 25 25 9 4	9 4
12 5 9 16 16 9 5	9 5
3 6 9 7 7 9 6	9 6
23 7 9 27 27 9 7	9 7
14 8 9 18 18 9 8	9 8
5 9 9 9 9 9 9	9 9
15 0 10 13 13 10 0	10 0
6 1 10 4 4 10 1	10 1
26 2 10 24 24 10 2	10 2
17 3 10 15 15 10 3	10 3
8 4 10 6 6 10 4	10 4
28 5 10 26 26 10 5	10 5
19 6 10 17 17 10 6	10 6
10 7 10 8 8 10 7	10 7
1 8 10 28 28 10 8	10 8
21 9 10 19 19 10 9	10 9
2 0 11 23 23 11 0	11 0
22 1 11 14 14 11 1	11 1
13 2 11 5 5 11 2	11 2
4 3 11 25 25 11 3	11 3
24 4 11 16 16 11 4	11 4
15 5 11 7 7 11 5	11 5
6 6 11 27 27 11 6	11 6
26 7 11 18 18 11 7	11 7
17 8 11 9 9 11 8	11 8
8 9 11 0 0 11 9	11 9
18 0 12 4 4 12 0	12 0
9 1 12 24 24 12 1	12 1
0 2 12 15 15 12 2	12 2
20 3 12 6 6 12 3	12 3
11 4 12 26 26 12 4	12 4
2 5 12 17 17 12 5	12 5
22 6 12 8 8 12 6	12 6

APÊNDICE B

TABELA ASCII COM OS CARACTERES IMPRIMÍVEIS

Este apêndice mostra a tabela com os caracteres ASCII imprimíveis usados como fonte de informação para os usuários 1 e 2.

Tabela B.1 – *Tabela ASCII dos caracteres imprimíveis*

Caractere	Decimal	Caractere	Decimal	Caractere	Decimal
Espaço	32	-	45	:	58
!	33	.	46	;	59
"	34	/	47	<	60
#	35	0	48	=	61
$	36	1	49	>	62
%	37	2	50	?	63
&	38	3	51	@	64
'	39	4	52	A	65
(	40	5	53	B	66
)	41	6	54	C	67
*	42	7	55	D	68
+	43	8	56	E	69
,	44	9	57	F	70
G	71	Z	90	m	109
H	72	[	91	n	110

Caractere	Decimal	Caractere	Decimal	Caractere	Decimal
I	73	\	92	o	111
J	74	]	93	p	112
K	75	^	94	q	113
L	76	_	95	r	114
M	77	`	96	s	115
N	78	a	97	t	116
O	79	b	98	u	117
P	80	c	99	v	118
Q	81	d	100	w	119
R	82	e	101	x	120
S	83	f	102	y	121
T	84	g	103	z	122
U	85	h	104	{	123
V	86	i	105	|	124
W	87	j	106	}	125
X	88	k	107	~	126
Y	89	l	108		

APÊNDICE C

PROGRAMAS UTILIZADOS NAS SIMULAÇÕES

```
%FONTE_ALEATORIA


% Rotina para Gerar Informação aleatória para os usuários e contabilizar
% número de erros por SNR no sistema.

SNR=0;
N=0;
n_int=input(' número de iterações por SNR        ')
snr_i=input(' Valor de SNR inicial          ')
snr_f=input(' Valor de SNR final          ')
QDE_ERROS=[1:1:snr_f];
FONTE='0123456789ABCDEFGHIJKLMNOPQRSTUWVXYZabcdefghijklmnopqrstuwvxyz
!"#$%&()*+,-./:;<=>?@[\]^_`{|}~';% Fonte ASCII de imprimíveis excluído
% o caracter " ' " devido limitação do comando no MATLAB.

for SNR=(snr_f:-2:snr_i)
    N_ERROS=0;
    QDE_ERROS(1,SNR)=0;

for N=1:n_int
    f1=FONTE(ceil(94.*rand(32,1)));%Informação aleatória do usúario 1.
    f2=FONTE(ceil(94.*rand(32,1)));%Informação aleatória do usúario 2.
    Codificador_ASCII;
    Decodificador_ASCII;
    CMP=not(eq([f1 f2],[saidausuario1 saidausuario2]));% Compara informação
%transmitida e recebida no sistema.
    N_ERROS=sum(CMP);
    QDE_ERROS(1,SNR)= QDE_ERROS(1,SNR)+ N_ERROS;% Gera vetor com número de
%erros contabilizados em cada SNR conforme posicionamento e valor de SNR
%inicial e final.
end
end
QDE_ERROS

%Codificador_ASCII


%Rotina para converter caracter ASCII no correspondente decimal.


clear 4dec_us1;
clear 32dec_us1;
4dec_us1=1:1:4;
clear 4dec_us2;
clear 32dec_us2;
4dec_us2=1:1:4;

for us=1:2
    if us==1
```

```
     f=f1;
    end
    if us==2
     f=f2;
    end
 for K=1:32

switch (f(K))
case ' ';m=32;
case '!';m=33;
case '"';m=34;
case '#';m=35;
case '$';m=36;
case '%';m=37;
case '&';m=38;
case '''';m=39;
case '(';m=40;
case ')';m=41;
case '*';m=42;
case '+';m=43;
case ',';m=44;
case '-';m=45;
case '.';m=46;
case '/';m=47;
case '0';m=48;
case '1';m=49;
case '2';m=50;
case '3';m=51;
case '4';m=52;
case '5';m=53;
case '6';m=54;
case '7';m=55;
case '8';m=56;
case '9';m=57;
case ':';m=58;
case ';';m=59;
case '<';m=60;
case '=';m=61;
case '>';m=62;
case '?';m=63;
case '@';m=64;
case 'A';m=65;
case 'B';m=66;
case 'C';m=67;
case 'D';m=68;
case 'E';m=69;
case 'F';m=70;
case 'G';m=71;
case 'H';m=72;
case 'I';m=73;
case 'J';m=74;
case 'K';m=75;
case 'L';m=76;
case 'M';m=77;
case 'N';m=78;
```

```
case 'O';m=79;
case 'P';m=80;
case 'Q';m=81;
case 'R';m=82;
case 'S';m=83;
case 'T';m=84;
case 'U';m=85;
case 'V';m=86;
case 'W';m=87;
case 'X';m=88;
case 'Y';m=89;
case 'Z';m=90;
case '[';m=91;
case '\';m=92;
case ']';m=93;
case '^';m=94;
case '_';m=95;
case '`';m=96;
case 'a';m=97;
case 'b';m=98;
case 'c';m=99;
case 'd';m=100;
case 'e';m=101;
case 'f';m=102;
case 'g';m=103;
case 'h';m=104;
case 'i';m=105;
case 'j';m=106;
case 'k';m=107;
case 'l';m=108;
case 'm';m=109;
case 'n';m=110;
case 'o';m=111;
case 'p';m=112;
case 'q';m=113;
case 'r';m=114;
case 's';m=115;
case 't';m=116;
case 'u';m=117;
case 'v';m=118;
case 'w';m=119;
case 'x';m=120;
case 'y';m=121;
case 'z';m=122;
case '{';m=123;
case '|';m=124;
case '}';m=125;
case '~';m=126;

      otherwise
       m=00;

end
    if us==1
```

```
     32dec_us1(K)=m;% Gera vetor com os respectivos 32 decimais da
informação %do usuário 1.
    end
    if us==2
     32dec_us2(K)=m;% Gera vetor com os respectivos 32 decimais da
informação %do usuário 2.
    end
t=[0];
 end
end

for K11=1:8 % Separação em 4 decimais para simulação (Simulador_2GAC);
    if K11==1
        4dec_us1(1:4)= 32dec_us1(1:4);
        4dec_us2(1:4)= 32dec_us2(1:4);
        sim('Simulador_2GAC');
    end
    if K11==2
        4dec_us1(1:4)= 32dec_us1(5:8);
        4dec_us2(1:4)= 32dec_us2(5:8);
        sim('Simulador_2GAC');

    end
    if K11==3
        4dec_us1(1:4)= 32dec_us1(9:12);
        4dec_us2(1:4)= 32dec_us2(9:12);
        sim('Simulador_2GAC');

    end
    if K11==4
        4dec_us1(1:4)= 32dec_us1(13:16);
        4dec_us2(1:4)= 32dec_us2(13:16);
        sim('Simulador_2GAC');

    end
    if K11==5
        4dec_us1(1:4)= 32dec_us1(17:20);
        4dec_us2(1:4)= 32dec_us2(17:20);
        sim('Simulador_2GAC');

    end
    if K11==6
        4dec_us1(1:4)= 32dec_us1(21:24);
        4dec_us2(1:4)= 32dec_us2(21:24);
        sim('Simulador_2GAC');

    end
    if K11==7
        4dec_us1(1:4)= 32dec_us1(25:28);
        4dec_us2(1:4)= 32dec_us2(25:28);
        sim('Simulador_2GAC');

    end
    if K11==8
```

```
        4dec_us1(1:4)= 32dec_us1(29:32);
        4dec_us2(1:4)= 32dec_us2(29:32);
        sim('Simulador_2GAC');

    end

end


%Codificador de Canal ( Codificador FOURIER).

%Bloco responsável pela codificação da informação enviada aos usuários.

function [X1, X2] = table(M1,M2)
x1=1:1:7;
x2=1:1:7;
X1=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];%Declaração de
%vetor (1x28)para armazenamento dos 4 códigos simultaneamente enviados ao
%usuário 1.
X2=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];%Declaração de
%vetor (1x28)para armazenamento dos 4 códigos simultaneamente enviados ao
%usuário 2.

for K=1:4;
switch (M1(K))
case 32;x1=[9 2 3 7 7 3 2];
case 33;x1=[ 14 3 3 3 3 3 3];
case 34;x1=[19 4 3 28 28 3 4];
case 35;x1=[24 5 3 24 24 3 5];
case 36;x1=[0 6 3 20 20 3 6];
case 37;x1=[5 7 3 16 16 3 7];
case 38;x1=[10 8 3 12 12 3 8];
case 39;x1=[15 9 3 8 8 3 9];
case 40;x1=[18 0 4 20 20 4 0];
case 41;x1=[23 1 4 16 16 4 1];
case 42;x1=[28 2 4 12 12 4 2];
case 43;x1=[4 3 4 8 8 4 3];
case 44;x1=[9 4 4 4 4 4 4];
case 45;x1=[14 5 4 0 0 4 5];
case 46;x1=[19 6 4 25 25 4 6];
case 47;x1=[24 7 4 21 21 4 7];
case 48;x1=[0 8 4 17 17 4 8];
case 49;x1=[5 9 4 13 13 4 9];
case 50;x1=[8 0 5 25 25 5 0];
case 51;x1=[13 1 5 21 21 5 1];
case 52;x1=[18 2 5 17 17 5 2];
case 53;x1=[23 3 5 13 13 5 3];
case 54;x1=[28 4 5 9 9 5 4];
case 55;x1=[4 5 5 5 5 5 5];
case 56;x1=[9 6 5 1 1 5 6];
case 57;x1=[14 7 5 26 26 5 7];
case 58;x1=[19 8 5 22 22 5 8];
case 59;x1=[24 9 5 18 18 5 9];
case 60;x1=[27 0 6 1 1 6 0];
```

```
case 61;x1=[3 1 6 26 26 6 1];
case 62;x1=[8 2 6 22 22 6 2];
case 63;x1=[13 3 6 18 18 6 3];
case 64;x1=[18 4 6 14 14 6 4];
case 65;x1=[23 5 6 10 10 6 5];
case 66;x1=[28 6 6 6 6 6 6];
case 67;x1=[4 7 6 2 2 6 7];
case 68;x1=[9 8 6 27 27 6 8];
case 69;x1=[14 9 6 23 23 6 9];
case 70;x1=[17 0 7 6 6 7 0];
case 71;x1=[22 1 7 2 2 7 1];
case 72;x1=[27 2 7  27 27 7 2];
case 73;x1=[3 3 7 23 23 7 3];
case 74;x1=[8 4 7 19 19 7 4];
case 75;x1=[13 5 7 15 15 7 5];
case 76;x1=[18 6 7 11 11 7 6];
case 77;x1=[23 7 7 7 7 7 7];
case 78;x1=[28 8 7 3 3 7 8];
case 79;x1=[4 9 7 28 28 7 9];
case 80;x1=[7 0 8 11 11 8 0];
case 81;x1=[12 1 8 7 7 8 1];
case 82;x1=[17 2 8 3 3 8 2];
case 83;x1=[22 3 8 28 28 8 3];
case 84;x1=[27 4 8 24 24 8 4];
case 85;x1=[3 5 8 20 20 8 5];
case 86;x1=[8 6 8 16 16 8 6];
case 87;x1=[13 7 8 12 12 8 7];
case 88;x1=[18 8 8 8 8 8 8];
case 89;x1=[23 9 8 4 4 8 9];
case 90;x1=[26 0 9 16 16 9 0];
case 91;x1=[2 1 9 12 12 9 1];
case 92;x1=[7 2 9 8 8 9 2];
case 93;x1=[12 3 9 4 4 9 3];
case 94;x1=[17 4 9 0 0 9 4];
case 95;x1=[22 5 9 25 25 9 5];
case 96;x1=[27 6 9 21 21 9 6];
case 97;x1=[3 7 9 17 17 9 7];
case 98;x1=[8 8 9 13 13 9 8];
case 99;x1=[13 9 9 9 9 9 9];
case 100;x1=[16 0 10 21 21 10 0];
case 101;x1=[21 1 10 17 17 10 1];
case 102;x1=[26 2 10 13 13 10 2];
case 103;x1=[2 3 10 9 9 10 3];
case 104;x1=[7 4 10 5 5 10 4];
case 105;x1=[12 5 10 1 1 10 5];
case 106;x1=[17 6 10 26 26 10 6];
case 107;x1=[22 7 10 22 22 10 7];
case 108;x1=[27 8 10 18 18 10 8];
case 109;x1=[3 9 10 14 14 10 9];
case 110;x1=[6 0 11 26 26 11 0];
case 111;x1=[11 1 11 22 22 11 1];
case 112;x1=[16 2 11 18 18 11 2];
case 113;x1=[21 3 11 14 14 11 3];
case 114;x1=[26 4 11 10 10 11 4];
case 115;x1=[2 5 11 6 6 11 5];
case 116;x1=[7 6 11 2 2 11 6];
```

```
case 117;x1=[12 7 11 27 27 11 7];
case 118;x1=[17 8 11 23 23 11 8];
case 119;x1=[22 9 11 19 19 11 9];
case 120;x1=[25 0 12 2 2 12 0];
case 121;x1=[1 1 12 27 27 12 1];
case 122;x1=[6 2 12 23 23 12 2];
case 123;x1=[11 3 12 19 19 12 3];
case 124;x1=[16 4 12 15 15 12 4];
case 125;x1=[21 5 12 11 11 12 5];
case 126;x1=[26 6 12 7 7 12 6];

end
    if K==1
        X1(1:7)=x1;% Código referente ao 1º caracter enviado ao usuário 1.
    end
    if K==2
        X1(8:14)=x1;% Código referente ao 2º caracter enviado ao usuário 1.
    end
    if K==3
        X1(15:21)=x1;% Código referente ao 3º caracter enviado ao usuário 1.
    end
    if K==4
        X1(22:28)=x1;% Código referente ao 4º caracter enviado ao usuário 1.
    end

end

 for K=1:4;
       switch (M2(K))
case  32;x2=[1 2 3 12 12 3 2];
case 33;x2=[ 21 3 3 3 3 3 3];
case 34;x2=[12 4 3 23 23 3 4];
case 35;x2=[3 5 3 14 14 3 5];
case 36;x2=[23 6 3 5 5 3 6];
case 37;x2=[14 7 3 25 25 3 7];
case 38;x2=[5 8 3 16 16 3 8];
case 39;x2=[25 9 3 7 7 3 9];
case 40;x2=[6 0 4 11 11 4 0];
case 41;x2=[26 1 4 2 2 4 1];
case 42;x2=[17 2 4 22 22 4 2];
case 43;x2=[8 3 4 13 13 4 3];
case 44;x2=[28 4 4 4 4 4 4];
case 45;x2=[19 5 4 24 24 4 5];
case 46;x2=[10 6 4 15 15 4 6];
case 47;x2=[1 7 4 6 6 4 7];
case 48;x2=[21 8 4 26 26 4 8];
case 49;x2=[12 9 4 17 17 4 9];
case 50;x2=[22 0 5 21 21 5 0];
case 51;x2=[13 1 5 12 12 5 1];
case 52;x2=[4 2 5 3 3 5 2];
case 53;x2=[24 3 5 23 23 5 3];
case 54;x2=[15 4 5 14 14 5 4];
case 55;x2=[6 5 5 5 5 5 5] ;
case 56;x2=[26 6 5 25 25 5 6];
case 57;x2=[17 7 5 16 16 5 7];
```

```
case 58;x2=[8 8 5 7 7 5 8];
case 59;x2=[28 9 5 27 27 5 9];
case 60;x2=[9 0 6 2 2 6 0];
case 61;x2=[0 1 6 22 22 6 1];
case 62;x2=[20 2 6 13 13 6 2];
case 63;x2=[11 3 6 4 4 6 3];
case 64;x2=[2 4 6 24 24 6 4];
case 65;x2=[22 5 6 15 15 6 5];
case 66;x2=[13 6 6 6 6 6 6];
case 67;x2=[4 7 6 26 26 6 7];
case 68;x2=[24 8 6 17 17 6 8];
case 69;x2=[15 9 6 8 8 6 9];
case 70;x2=[25 0 7 12 12 7 0];
case 71;x2=[16 1 7 3 3 7 1];
case 72;x2=[7 2 7 23 23 7 2];
case 73;x2=[27 3 7 14 14 7 3];
case 74;x2=[18 4 7 5 5 7 4];
case 75;x2=[9 5 7 25 25 7 5];
case 76;x2=[0 6 7 16 16 7 6];
case 77;x2=[20 7 7 7 7 7 7];
case 78;x2=[11 8 7 27 27 7 8];
case 79;x2=[2 9 7 18 18 7 9];
case 80;x2=[12 0 8 22 22 8 0];
case 81;x2=[3 1 8 13 13 8 1];
case 82;x2=[23 2 8 4 4 8 2];
case 83;x2=[14 3 8 24 24 8 3];
case 84;x2=[5 4 8 15 15 8 4];
case 85;x2=[25 5 8 6 6 8 5];
case 86;x2=[16 6 8 26 26 8 6];
case 87;x2=[7 7 8 17 17 8 7];
case 88;x2=[27 8 8 8 8 8 8];
case 89;x2=[18 9 8 28 28 8 9];
case 90;x2=[28 0 9 3 3 9 0];
case 91;x2=[19 1 9 23 23 9 1];
case 92;x2=[10 2 9 14 14 9 2];
case 93;x2=[1 3 9 5 5 9 3];
case 94;x2=[21 4 9 25 25 9 4];
case 95;x2=[12 5 9 16 16 9 5];
case 96;x2=[3 6 9 7 7 9 6];
case 97;x2=[23 7 9 27 27 9 7];
case 98;x2=[14 8 9 18 18 9 8];
case 99;x2=[5 9 9 9 9 9 9];
case 100;x2=[15 0 10 13 13 10 0];
case 101;x2=[6 1 10 4 4 10 1];
case 102;x2=[26 2 10 24 24 10 2];
case 103;x2=[17 3 10 15 15 10 3];
case 104;x2=[8 4 10 6 6 10 4];
case 105;x2=[28 5 10 26 26 10 5];
case 106;x2=[19 6 10 17 17 10 6];
case 107;x2=[10 7 10 8 8 10 7];
case 108;x2=[1 8 10 28 28 10 8];
case 109;x2=[21 9 10 19 19 10 9];
case 110;x2=[2 0 11 23 23 11 0];
case 111;x2=[22 1 11 14 14 11 1];
case 112;x2=[13 2 11 5 5 11 2];
case 113;x2=[4 3 11 25 25 11 3];
```

```
case 114;x2=[24 4 11 16 16 11 4];
case 115;x2=[15 5 11 7 7 11 5];
case 116;x2=[6 6 11 27 27 11 6];
case 117;x2=[26 7 11 18 18 11 7];
case 118;x2=[17 8 11 9 9 11 8];
case 119;x2=[8 9 11 0 0 11 9];
case 120;x2=[18 0 12 4 4 12 0];
case 121;x2=[9 1 12 24 24 12 1];
case 122;x2=[0 2 12 15 15 12 2];
case 123;x2=[20 3 12 6 6 12 3];
case 124;x2=[11 4 12 26 26 12 4];
case 125;x2=[2 5 12 17 17 12 5];
case 126;x2=[22 6 12 8 8 12 6];


       end
    if K==1
        X2(1:7)=x2;% Código referente ao 1º caracter enviado ao usuário 2.
    end
    if K==2
        X2(8:14)=x2;% Código referente ao 2º caracter enviado ao usuário 2.
    end
    if K==3
        X2(15:21)=x2;% Código referente ao 3º caracter enviado ao usuário 2.
    end
    if K==4
        X2(22:28)=x2;% Código referente ao 4º caracter enviado ao usuário 2.
    end

 end
end

% Transformada Numérica de FOURIER (TNF)


function Y = TNF(y)

% Bloco funcional responsável pela Transformada Numérica de Fourier das
%autossequencias recebidas.
Y= [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];%Declaração de
%vetor (1x28)para armazenamento de Y ( somatório de 4 autossequencias de
%usuário por transmissão).

F=[24 24 24 24 24 24 24; % Matriz F
   24 23 16 25 1 7 20;
   24 16 1 20 23 25 7;
   24 25 20 16 7 23 1;
   24 1 23 7 16 20 25;
   24 7 25 23 20 1 16;
   24 20 7 1 25 16 23];


Y_parc=y*F; % Transformada Numérica de Fourier de y
```

```
Ys(1:7)= Y_parc(1,1:7);
Ys(8:14)= Y_parc(2,1:7);
Ys(15:21)= Y_parc(3,1:7);
Ys(22:28)= Y_parc(4,1:7);
Y=mod(Y,29);

      end

% ALGORITMO DE DECODIFICAÇÃO.

%r1-vetor (1x28) contendo as prováveis autossequências enviadas ao usuário
1.
%r2-vetor (1x28) contendo as prováveis autossequências enviadas ao usuário
2.

function [rc_x1,rc_x2]  = corrector(r1,r2)
% Bloco funcional responsável pela detecção e correção de até dois erros nas
autossequencias recebidas.

eml.extrinsic('cong');
rc1=[1:1:7;1:1:7;1:1:7;1:1:7];
rc2=[1:1:7;1:1:7;1:1:7;1:1:7];
r_testx1=(1:7);% Memória contendo autossequências em teste do usuário 1.
r_testx2=(1:7);% Memória contendo autossequências em teste do usuário 2.
r1x1=(1:7);% autosequência em análise do usuário 1.
r2x2=(1:7);% autosequência em análise do usuário 2.
r=[1:1:7];

% DECISÃO ABRUPTA PARA O USUÁRIO 1.
for h=1:7
    if r1(h)>=28.5%Se houver algum número maior ou igual a 28,5 na
%autosequência este receberá 28(limite superior).
        r1(h)=28;
    end
end
h=0;
for h=1:7
    if r1(h)<0%Se houver algum número negativo na autosequência este
receberá %zero(limite inferior).
        r1(h)=0;
    end
end

% ARREDONDAMENTO DAS AUTOSSEQUÊNCIAS DO USUÁRIO 1.
h=0;
r1=round(r1);
r1=mod(r1,29);

% DECISÃO ABRUPTA PARA O USUÁRIO 2.
for h=1:7
    if r2(h)>=28.5%Se houver algum número maior ou igual a 28,5 na
%autosequência este receberá 28(limite superior).
        r2(h)=28;
    end
```

```
end
for h=1:7
    if r2(h)<0%Se houver algum número negativo na autosequência este
receberá %zero(limite inferior).
        r2(h)=0;
    end
end

% ARREDONDAMENTO DAS AUTOSSEQUÊNCIAS DO USUÁRIO 2.

r2=round(r2);
r2=mod(r2,29);

% ANÁLISES DAS AUTOSSEQUÊNCIAS DO USUÁRIO 1.

for l=1:4
    if l==1
        r1x1(1:7)=r1(1:7);
    end
    if l==2
        r1x1(1:7)=r1(8:14);
    end
    if l==3
        r1x1(1:7)=r1(15:21);
    end
    if l==4
        r1x1(1:7)=r1(22:28);
    end
% r1x1=[ro r1 r2 r3 r4 r5 r6] - autosequência do usuário 1 em análise
rc1(l,1:7)=r1x1;
S=r1x1*F;
S=mod(S,29);

% CÁLCULO DA SÍNDROME USUÁRIO 1.
if all((S-r1x1)==0);%Verifica se autosequência analisada é palavra código
    rc1(l,1:7)=r1x1;

% CORREÇÃO DE UM ERRO USUÁRIO 1 – SIMÉTRICO.
else
    if r1x1(2)==r1x1(7)&& r1x1(3)==r1x1(6)&& r1x1(4)==r1x1(5)%Verifica se
apenas r0 está errado.
        rc1(l,1:7)=r1x1;
        r0calc=cong([r1x1],29,1);
        r_testx1(1)=mod(r0calc,29);
        r_testx1(2:7)=r1x1(2:7);
        s=r_testx1*F;
        s=mod(s,29);
            if all ((r_testx1-s)==0);
            rc1(l,1:7)=r_testx1;

% CORREÇÃO DE UM ERRO USUÁRIO 1 – NÃO SIMÉTRICO
            else
                kk=0;
              for kk=(1:3);
                  if kk==1;%Verifica se r1 ou r6 errado.
```

```
                    rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(3)+r1x1(4)+r1x1(5)+r1x1(6)]);
                    rcalc=mod(rcalc,29);
                    r_testx1=[r1x1(1) rcalc r1x1(3) r1x1(4) r1x1(5) r1x1(6)
rcalc];
                    s1=r_testx1*F;
                    s1=mod(s1,29);
                            if all ((s1-r_testx1)==0);
                            rc1(l,1:7)=r_testx1;
                            end
                    end
                    if kk==2;%Verifica se r2 ou r5 errado.
                    rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(2)+r1x1(4)+r1x1(5)+r1x1(7)]);
                    rcalc=mod(rcalc,29);
                    r_testx1=[r1x1(1) r1x1(2) rcalc  r1x1(4) r1x1(5)  rcalc
r1x1(7)];
                    s1=r_testx1*F;
                    s1=mod(s1,29);
                            if all ((s1-r_testx1)==0);
                            rc1(l,1:7)=r_testx1;
                            end

                    end
                    if kk==3;%Verifica se r3 ou r4 errado.
                    rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(2)+r1x1(3)+r1x1(6)+r1x1(7)]);
                    rcalc=mod(rcalc,29);
                    r_testx1=[r1x1(1) r1x1(2) r1x1(3) rcalc rcalc r1x1(6)
r1x1(7)];
                    s1=r_testx1*F;
                    s1=mod(s1,29);
                           if all ((s1-r_testx1)==0);
                            rc1(l,1:7)=r_testx1;
                            end

                    end
                end
              end

% CORREÇÃO DOIS ERROS USUÁRIO 1 - SIMÉTRICO E NÃO SIMÉTRICO
    else
                        if r1x1(2)~=r1x1(7)&& r1x1(3)==r1x1(6)&&
r1x1(4)==r1x1(5);%Verifica se r0 e r6 errados.
                             r_subs=r1x1(2);
                             r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r1x1(5)
r1x1(6) r_subs];
                             r0calc=cong([r_subsx1],29,1);
                             r_testx1(1)=mod(r0calc,29);
                             r_testx1(2:7)=r_subsx1(2:7);
                             s=r_testx1*F;
                             s=mod(s,29);
                         if all ((r_testx1-s)==0);
                                 rc1(l,1:7)=r_testx1;
                         else   %Verifica se r0 e r1 errados.
```

```
                            r_subs=r1x1(7);
                            r_subsx1=[r1x1(1) r_subs r1x1(3) r1x1(4) r1x1(5)
r1x1(6) r1x1(7)];
                            r0calc=cong([r_subsx1],29,1);
                            r_testx1(1)=mod(r0calc,29);
                            r_testx1(2:7)=r_subsx1(2:7);
                            s=r_testx1*F;
                            s=mod(s,29);
                            if all ((r_testx1-s)==0);
                                rc1(l,1:7)=r_testx1;
                            else %Verifica se r1 e r6 errados.
                                rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(3)+r1x1(4)+r1x1(5)+r1x1(6)]);
                                rcalc=mod(rcalc,29);
                                r_testx1=[r1x1(1) rcalc r1x1(3) r1x1(4)
r1x1(5) r1x1(6) rcalc];
                                s1=r_testx1*F;
                                s1=mod(s1,29);
                                   if all ((s1-r_testx1)==0);
                                      rc1(l,1:7)=r_testx1;
                                   end
                           end
                         end
                        end

                        if r1x1(2)==r1x1(7)&& r1x1(3)~=r1x1(6)&&
r1x1(4)==r1x1(5);%Verifica se r0 e r5 errados.
                            r_subs=r1x1(3);
                            r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r1x1(5)
r_subs r1x1(7)];
                            r0calc=cong([r_subsx1],29,1);
                            r_testx1(1)=mod(r0calc,29);
                            r_testx1(2:7)=r_subsx1(2:7);
                            s=r_testx1*F;
                            s=mod(s,29);
                         if all ((r_testx1-s)==0);
                                rc1(l,1:7)=r_testx1;
                         else %Verifica se r0 e r2 errados.
                            r_subs=r1x1(6);
                            r_subsx1=[r1x1(1) r1x1(2) r_subs  r1x1(4) r1x1(5)
r1x1(6) r1x1(7)];
                            r0calc=cong([r_subsx1],29,1);
                            r_testx1(1)=mod(r0calc,29);
                            r_testx1(2:7)=r_subsx1(2:7);
                            s=r_testx1*F;
                            s=mod(s,29);
                            if all ((r_testx1-s)==0);
                                rc1(l,1:7)=r_testx1;
                            else %Verifica se r2 e r5 errados.
                                rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(2)+r1x1(4)+r1x1(5)+r1x1(7)]);
                                rcalc=mod(rcalc,29);
                                r_testx1=[r1x1(1) r1x1(2) rcalc  r1x1(4)
r1x1(5)  rcalc r1x1(7)];
                                s1=r_testx1*F;
```

```
                                s1=mod(s1,29);
                                   if all ((s1-r_testx1)==0);
                                      rc1(l,1:7)=r_testx1;
                                   end
                             end
                          end
                         end

                          if r1x1(2)==r1x1(7)&& r1x1(3)==r1x1(6)&&
r1x1(4)~=r1x1(5);%Verifica se r0 e r4 errados.
                              r_subs=r1x1(4);
                              r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r_subs
r1x1(6) r1x1(7)];
                              r0calc=cong([r_subsx1],29,1);
                              r_testx1(1)=mod(r0calc,29);
                              r_testx1(2:7)=r_subsx1(2:7);
                              s=r_testx1*F;
                              s=mod(s,29);
                          if all ((r_testx1-s)==0);
                                  rc1(l,1:7)=r_testx1;
                          else %Verifica se r0 e r3 errados.
                              r_subs=r1x1(5);
                              r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r_subs r1x1(5)
r1x1(6) r1x1(7)];
                              r0calc=cong([r_subsx1],29,1);
                              r_testx1(1)=mod(r0calc,29);
                              r_testx1(2:7)=r_subsx1(2:7);
                              s=r_testx1*F;
                              s=mod(s,29);
                              if all ((r_testx1-s)==0);
                                  rc1(l,1:7)=r_testx1;
                              else %Verifica se r3 e r4 errados.
                                  rcalc=1/2*(1*23*r1x1(1)-
[r1x1(1)+r1x1(2)+r1x1(3)+r1x1(6)+r1x1(7)]);
                                  rcalc=mod(rcalc,29);
                                  r_testx1=[r1x1(1) r1x1(2) r1x1(3) rcalc rcalc
r1x1(6) r1x1(7)];
                                  s1=r_testx1*F;
                                  s1=mod(s1,29);
                                     if all ((s1-r_testx1)==0);
                                     rc1(l,1:7)=r_testx1;
                                     end
                              end
                          end
                         end
                        if r1x1(2)~=r1x1(7)&& r1x1(3)~=r1x1(6)&&
r1x1(4)==r1x1(5);%Verifica se r5 e r6 errados.
                              r_subs1=r1x1(2);
                              r_subs2=r1x1(3);
                              r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r1x1(5)
r_subs2 r_subs1];
                              s=r_subsx1*F;
                              s=mod(s,29);
                          if all ((r_subsx1-s)==0);
                                  rc1(l,1:7)=r_subsx1;
```

```
                        else %Verifica se r2 e r6 errados.
                            r_subs1=r1x1(2);
                            r_subs2=r1x1(6);
                            r_subsx1=[r1x1(1) r1x1(2) r_subs2 r1x1(4) r1x1(5)
r1x1(6) r_subs1];
                            s=r_subsx1*F;
                            s=mod(s,29);
                            if all ((r_subsx1-s)==0);
                                rc1(l,1:7)=r_subsx1;
                                  else %Verifica se r1 e r5 errados.
                                    r_subs1=r1x1(7);
                                    r_subs2=r1x1(3);
                                    r_subsx1=[r1x1(1) r_subs1 r1x1(3) r1x1(4)
r1x1(5) r_subs2 r1x1(7)];
                                    s=r_subsx1*F;
                                    s=mod(s,29);
                                    if all ((r_subsx1-s)==0);
                                      rc1(l,1:7)=r_subsx1;
                                    else %Verifica se r1 e r2 errados.
                                      r_subs1=r1x1(7);
                                      r_subs2=r1x1(6);
                                      r_subsx1=[r1x1(1) r_subs1 r_subs2
r1x1(4) r1x1(5) r1x1(6) r1x1(7)];
                                      s=r_subsx1*F;
                                      s=mod(s,29);
                                      if all ((r_subsx1-s)==0);
                                      rc1(l,1:7)=r_subsx1;
                                      end

                                    end
                            end
                          end
                        end
                        if r1x1(2)~=r1x1(7)&& r1x1(3)==r1x1(6)&&
r1x1(4)~=r1x1(5);%Verifica se r4 e r6 errados.
                            r_subs1=r1x1(2);
                            r_subs2=r1x1(4);
                            r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r_subs2
r1x1(6) r_subs1];
                            s=r_subsx1*F;
                            s=mod(s,29);
                        if all ((r_subsx1-s)==0);
                                rc1(l,1:7)=r_subsx1;
                        else %Verifica se r3 e r6 errados.
                            r_subs1=r1x1(2);
                            r_subs2=r1x1(5);
                            r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r_subs2
r1x1(5) r1x1(6) r_subs1];
                            s=r_subsx1*F;
                            s=mod(s,29);
                            if all ((r_subsx1-s)==0);
                                rc1(l,1:7)=r_subsx1;
                                  else %Verifica se r1 e r4 errados.
                                    r_subs1=r1x1(7);
                                    r_subs2=r1x1(4);
```

```
                                    r_subsx1=[r1x1(1) r_subs1 r1x1(3) r1x1(4)
r_subs2 r1x1(6) r1x1(7)];
                                    s=r_subsx1*F;
                                    s=mod(s,29);
                                    if all ((r_subsx1-s)==0);
                                      rc1(l,1:7)=r_subsx1;
                                    else %Verifica se r1 e r3 errados.
                                      r_subs1=r1x1(7);
                                      r_subs2=r1x1(5);
                                      r_subsx1=[r1x1(1) r_subs1 r1x1(3)
r_subs2 r1x1(5) r1x1(6) r1x1(7)];
                                      s=r_subsx1*F;
                                      s=mod(s,29);
                                      if all ((r_subsx1-s)==0);
                                      rc1(l,1:7)=r_subsx1;
                                      end

                                    end
                              end
                          end
                        end
                        if r1x1(2)==r1x1(7)&& r1x1(3)~=r1x1(6)&&
r1x1(4)~=r1x1(5);%Verifica se r4 e r5 errados.
                              r_subs1=r1x1(4);
                              r_subs2=r1x1(3);
                              r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r1x1(4) r_subs1
r_subs2 r1x1(7) ];
                              s=r_subsx1*F;
                              s=mod(s,29);
                          if all ((r_subsx1-s)==0);
                                  rc1(l,1:7)=r_subsx1;
                          else %Verifica se r2 e r4 errados.
                              r_subs1=r1x1(4);
                              r_subs2=r1x1(6);
                              r_subsx1=[r1x1(1) r1x1(2) r_subs2 r1x1(4) r_subs1
r1x1(6) r1x1(7)];
                              s=r_subsx1*F;
                              s=mod(s,29);
                              if all ((r_subsx1-s)==0);
                                  rc1(l,1:7)=r_subsx1;
                                else %Verifica se r3 e r5 errados.
                                  r_subs1=r1x1(5);
                                  r_subs2=r1x1(3);
                                  r_subsx1=[r1x1(1) r1x1(2) r1x1(3) r_subs1
r1x1(5) r_subs2 r1x1(7)];
                                  s=r_subsx1*F;
                                  s=mod(s,29);
                                  if all ((r_subsx1-s)==0);
                                    rc1(l,1:7)=r_subsx1;
                                  else %Verifica se r2 e r3 errados.
                                    r_subs1=r1x1(5);
                                    r_subs2=r1x1(6);
                                    r_subsx1=[r1x1(1) r1x1(2) r_subs2
r_subs1 r1x1(5) r1x1(6) r1x1(7)];
                                    s=r_subsx1*F;
```

```
                                        s=mod(s,29);
                                        if all ((r_subsx1-s)==0);
                                        rc1(l,1:7)=r_subsx1;
                                        end

                                    end
                                end
                            end
                        end

    end
end

 % ANÁLISES DAS AUTOSSEQUÊNCIAS DO USUÁRIO 2.

    if l==1
        r2x2(1:7)=r2(1:7);
    end
    if l==2
        r2x2(1:7)=r2(8:14);
    end
    if l==3
        r2x2(1:7)=r2(15:21);
    end
    if l==4
        r2x2(1:7)=r2(22:28);
    end
% r2x2=[ro r1 r2 r3 r4 r5 r6] - autosequência do usuário 2 em análise
rc2(l,1:7)=r2x2;
S=r2x2*F;
S=mod(S,29);
Sr2x2=mod(S+r2x2,29);

% CÁLCULO DA SÍNDROME USUÁRIO 2.
if all((Sr2x2)==0);%Verifica se autosequência analisada é palavra código
    rc2(l,1:7)=r2x2;

% CORREÇÃO DE UM ERRO USUÁRIO 2 – SIMÉTRICO.
else
    if r2x2(2)==r2x2(7)&& r2x2(3)==r2x2(6)&& r2x2(4)==r2x2(5)%Verifica se
apenas r0 está errado.
     rc2(l,1:7)=r2x2;
     r0calc=cong([r2x2],29,-1);
     r_testx2(1)=mod(r0calc,29);
     r_testx2(2:7)=r2x2(2:7);

     s=r_testx2*F;
     s=mod(s,29);
     a=(mod(-r_testx2,29));
           if all((a-s)==0);
           rc2(l,1:7)=r_testx2;
           else
                kk=0;
                for kk=(1:3);
                     if kk==1;%Verifica se r1 ou r6 errados.
```

```
                    rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(3)+r2x2(4)+r2x2(5)+r2x2(6)]);
                    rcalc=mod(rcalc,29);
                    r_testx2=[r2x2(1) rcalc r2x2(3) r2x2(4) r2x2(5) r2x2(6)
rcalc];
                    s2=r_testx2*F;
                    s2=mod(s2,29);
                    a=(mod(-r_testx2,29)) ;
                        if all ((s2-a)==0);
                        rc2(l,1:7)=r_testx2;
                        end
                    end
                    if kk==2;%Verifica se r2 ou r5 errados.
                    rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(2)+r2x2(4)+r2x2(5)+r2x2(7)]);
                    rcalc=mod(rcalc,29);
                    r_testx2=[r2x2(1) r2x2(2) rcalc  r2x2(4) r2x2(5)  rcalc
r2x2(7)];
                    s2=r_testx2*F;
                    s2=mod(s2,29);
                    a=(mod(-r_testx2,29));
                          if all ((s2-a)==0);
                          rc2(l,1:7)=r_testx2;
                          end

                    end
                    if kk==3;%Verifica se r3 ou r4 errados.
                    rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(2)+r2x2(3)+r2x2(6)+r2x2(7)]);
                    rcalc=mod(rcalc,29);
                    r_testx2=[r2x2(1) r2x2(2) r2x2(3) rcalc rcalc r2x2(6)
r2x2(7)];
                    s2=r_testx2*F;
                    s2=mod(s2,29);
                    a=(mod(-r_testx2,29));
                             if all ((s2-a)==0);
                             rc2(l,1:7)=r_testx2;
                             end

                    end
                end
            end

% CORREÇÃO DOIS ERROS USUÁRIO 2 – SIMÉTRICO E NÃO SIMÉTRICO.
            else
                if r2x2(2)~=r2x2(7)&& r2x2(3)==r2x2(6)&&
r2x2(4)==r2x2(5);%Verifica se r0 e r6 errados.
                          r_subs=r2x2(2);
                          r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r2x2(5)
r2x2(6) r_subs];
                          r0calc=cong([r_subsx1],29,-1);
                          r_testx2(1)=mod(r0calc,29);
                          r_testx2(2:7)=r_subsx1(2:7);
                          s2=r_testx2*F;
                          s2=mod(s2,29);
```

```
                            a=(mod(-r_testx2,29)) ;
                              if all ((s2-a)==0);
                              rc2(l,1:7)=r_testx2;
                              else    %Verifica se r0 e r1 errados.
                            r_subs=r2x2(7);
                            r_subsx1=[r2x2(1) r_subs r2x2(3) r2x2(4) r2x2(5)
r2x2(6) r2x2(7)];
                            r0calc=cong([r_subsx1],29,-1);
                            r_testx2(1)=mod(r0calc,29);
                            r_testx2(2:7)=r_subsx1(2:7);
                            s2=r_testx2*F;
                            s2=mod(s2,29);
                            a=(mod(-r_testx2,29)) ;
                            if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                            else %Verifica se r1 e r6 errados.
                                rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(4)+r2x2(3)+r2x2(6)+r2x2(5)]);
                                rcalc=mod(rcalc,29);
                                r_testx2=[r2x2(1) rcalc r2x2(3) r2x2(4)
r2x2(5) r2x2(6) rcalc];
                                s2=r_testx2*F;
                                s2=mod(s2,29);
                                a=(mod(-r_testx2,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                                end
                           end
                         end
                        end
                        if r2x2(2)==r2x2(7)&& r2x2(3)==r2x2(6)&&
r2x2(4)~=r2x2(5);%Verifica se r0 e r4 errados.
                            r_subs=r2x2(4);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r_subs
r2x2(6) r2x2(7)];
                            r0calc=cong([r_subsx1],29,-1);
                            r_testx2(1)=mod(r0calc,29);
                            r_testx2(2:7)=r_subsx1(2:7);
                            s2=r_testx2*F;
                            s2=mod(s2,29);
                            a=(mod(-r_testx2,29)) ;
                         if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                         else %Verifica se r0 e r3 errados.
                            r_subs=r2x2(5);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r_subs r2x2(5)
r2x2(6) r2x2(7)];
                            r0calc=cong([r_subsx1],29,-1);
                            r_testx2(1)=mod(r0calc,29);
                            r_testx2(2:7)=r_subsx1(2:7);
                            s2=r_testx2*F;
                            s2=mod(s2,29);
                            a=(mod(-r_testx2,29)) ;
                            if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                            else %Verifica se r3 e r4 errados.
```

```
                                rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(2)+r2x2(3)+r2x2(6)+r2x2(7)]);
                                rcalc=mod(rcalc,29);
                                r_testx2=[r2x2(1) r2x2(2) r2x2(3) rcalc rcalc
r2x2(6) r2x2(7)];
                                s2=r_testx2*F;
                                s2=mod(s2,29);
                                a=(mod(-r_testx2,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                                end
                            end
                         end
                        end
                        if r2x2(2)==r2x2(7)&& r2x2(3)~=r2x2(6)&&
r2x2(4)==r2x2(5);%Verifica se r0 e r5 errados.
                            r_subs=r2x2(3);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r2x2(5)
r_subs r2x2(7)];
                            r0calc=cong([r_subsx1],29,-1);
                            r_testx2(1)=mod(r0calc,29);
                            r_testx2(2:7)=r_subsx1(2:7);
                            s2=r_testx2*F;
                            s2=mod(s2,29);
                            a=(mod(-r_testx2,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                                else %Verifica se r0 e r2 errados.
                            r_subs=r2x2(6);
                            r_subsx1=[r2x2(1) r2x2(2) r_subs  r2x2(4) r2x2(5)
r2x2(6) r2x2(7)];
                            r0calc=cong([r_subsx1],29,-1);
                            r_testx2(1)=mod(r0calc,29);
                            r_testx2(2:7)=r_subsx1(2:7);
                            s2=r_testx2*F;
                            s2=mod(s2,29);
                            a=(mod(-r_testx2,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                                else %Verifica se r2 e r5 errados.
                                rcalc=1/2*((-1)*23*r2x2(1)-
[r2x2(1)+r2x2(2)+r2x2(4)+r2x2(5)+r2x2(7)]);
                                rcalc=mod(rcalc,29);
                                r_testx2=[r2x2(1) r2x2(2) rcalc  r2x2(4)
r2x2(5)  rcalc r2x2(7)];
                                s2=r_testx2*F;
                                s2=mod(s2,29);
                                a=(mod(-r_testx2,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_testx2;
                                end
                            end
                          end
                        end
                        if r2x2(2)~=r2x2(7)&& r2x2(3)~=r2x2(6)&&
r2x2(4)==r2x2(5);%Verifica se r5 e r6 errados.
```

```
                            r_subs1=r2x2(2);
                            r_subs2=r2x2(3);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r2x2(5)
r_subs2 r_subs1];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
                            a=(mod(-r_subsx1,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_subsx1;
                                else %Verifica se r2 e r6 errados.
                            r_subs1=r2x2(2);
                            r_subs2=r2x2(6);
                            r_subsx1=[r2x2(1) r2x2(2) r_subs2 r2x2(4) r2x2(5)
r2x2(6) r_subs1];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
                            a=(mod(-r_subsx1,29)) ;
                            if all ((s2-a)==0);
                                rc2(l,1:7)=r_subsx1;
                                  else %Verifica se r1 e r5 errados.
                                    r_subs1=r2x2(7);
                                    r_subs2=r2x2(3);
                                    r_subsx1=[r2x2(1) r_subs1 r2x2(3) r2x2(4)
r2x2(5) r_subs2 r2x2(7)];
                                    s2=r_subsx1*F;
                                    s2=mod(s2,29);
                                    a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      else %Verifica se r1 e r2 errados.
                                      r_subs1=r2x2(7);
                                      r_subs2=r2x2(6);
                                      r_subsx1=[r2x2(1) r_subs1 r_subs2
r2x2(4) r2x2(5) r2x2(6) r2x2(7)];
                                      s2=r_subsx1*F;
                                      s2=mod(s2,29);
                                      a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      end

                                    end
                            end
                          end
                        end
                        if r2x2(2)~=r2x2(7)&& r2x2(3)==r2x2(6)&&
r2x2(4)~=r2x2(5);%Verifica se r4 e r6 errados.
                            r_subs1=r2x2(2);
                            r_subs2=r2x2(4);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r_subs2
r2x2(6) r_subs1];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
                            a=(mod(-r_subsx1,29)) ;
                                if all ((s2-a)==0);
```

```
                                rc2(l,1:7)=r_subsx1;
                                else %Verifica se r3 e r6 errados.
                            r_subs1=r2x2(2);
                            r_subs2=r2x2(5);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r_subs2 r2x2(5)
r2x2(6) r_subs1];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
                            a=(mod(-r_subsx1,29)) ;
                            if all ((s2-a)==0);
                                rc2(l,1:7)=r_subsx1;
                                  else %Verifica se r1 e r4 errados.
                                    r_subs1=r2x2(7);
                                    r_subs2=r2x2(4);
                                    r_subsx1=[r2x2(1) r_subs1 r2x2(3) r2x2(4)
r_subs2 r2x2(6) r2x2(7)];
                                    s2=r_subsx1*F;
                                    s2=mod(s2,29);
                                    a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      else %Verifica se r1 e r3 errados.
                                      r_subs1=r2x2(7);
                                      r_subs2=r2x2(5);
                                      r_subsx1=[r2x2(1) r_subs1 r2x2(3)
r_subs2 r2x2(5) r2x2(6) r2x2(7)];
                                      s2=r_subsx1*F;
                                      s2=mod(s2,29);
                                      a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      end

                                    end
                            end
                          end
                        end
                        if r2x2(2)==r2x2(7)&& r2x2(3)~=r2x2(6)&&
r2x2(4)~=r2x2(5);%Verifica se r4 e r5 errados.
                            r_subs1=r2x2(4);
                            r_subs2=r2x2(3);
                            r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r2x2(4) r_subs1
r_subs2 r2x2(7)];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
                            a=(mod(-r_subsx1,29)) ;
                                if all ((s2-a)==0);
                                rc2(l,1:7)=r_subsx1;
                                else %Verifica se r2 e r4 errados.
                            r_subs1=r2x2(4);
                            r_subs2=r2x2(6);
                            r_subsx1=[r2x2(1) r2x2(2) r_subs2 r2x2(4) r_subs1
r2x2(6) r2x2(7)];
                            s2=r_subsx1*F;
                            s2=mod(s2,29);
```

```
                            a=(mod(-r_subsx1,29)) ;
                            if all ((s2-a)==0);
                                rc2(l,1:7)=r_subsx1;
                                  else %Verifica se r3 e r5 errados.
                                    r_subs1=r2x2(5);
                                    r_subs2=r2x2(3);
                                    r_subsx1=[r2x2(1) r2x2(2) r2x2(3) r_subs1
r2x2(5) r_subs2 r2x2(7)];
                                    s2=r_subsx1*F;
                                    s2=mod(s2,29);
                                    a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      else %Verifica se r2 e r3 errados.
                                      r_subs1=r2x2(5);
                                      r_subs2=r2x2(6);
                                      r_subsx1=[r2x2(1) r2x2(2) r_subs2
r_subs1 r2x2(5) r2x2(6) r2x2(7)];
                                      s2=r_subsx1*F;
                                      s2=mod(s2,29);
                                      a=(mod(-r_subsx1,29)) ;
                                      if all ((s2-a)==0);
                                      rc2(l,1:7)=r_subsx1;
                                      end

                                    end
                            end
                        end
                    end

    end
end
end

rc_x1=[[rc1(1,1:7)] [rc1(2,1:7)] [rc1(3,1:7)] [rc1(4,1:7)]];%
Autossequencias de saida do usuário 1.
rc_x2=[[rc2(1,1:7)] [rc2(2,1:7)] [rc2(3,1:7)] [rc2(4,1:7)]];%
Autossequencias de saida do usuário 2.
end

%Decodificado de canal (Decodificador FOURIER)

function [S1,S2] = table(X1,X2)
%Bloco funcional responsável pela decodificação de canal.

xtab=[1 2 3 4 5 6 7];
X1_F=[1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1];
X2_F=[1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1;1 1 1 1 1 1 1];
S1=[32 32 32 32];
S2=[32 32 32 32];
x1=1:1:7;
x11=1:1:7;
x2=1:1:7;
x22=1:1:7;
M1=0;
```

```
M2=0;
dcal=0;

%Decodificado de canal do usuário 1

%Declaração das palavras códigos do usuário 1.
X=[9 2 3 7 7 3 2;
14 3 3 3 3 3 3;
19 4 3 28 28 3 4;
24 5 3 24 24 3 5;
0 6 3 20 20 3 6;
5 7 3 16 16 3 7;
10 8 3 12 12 3 8;
15 9 3 8 8 3 9;
18 0 4 20 20 4 0;
23 1 4 16 16 4 1;
28 2 4 12 12 4 2;
4 3 4 8 8 4 3;
9 4 4 4 4 4 4;
14 5 4 0 0 4 5;
19 6 4 25 25 4 6;
24 7 4 21 21 4 7;
0 8 4 17 17 4 8;
5 9 4 13 13 4 9;
8 0 5 25 25 5 0;
13 1 5 21 21 5 1;
18 2 5 17 17 5 2;
23 3 5 13 13 5 3;
28 4 5 9 9 5 4;
4 5 5 5 5 5 5;
9 6 5 1 1 5 6;
14 7 5 26 26 5 7;
19 8 5 22 22 5 8;
24 9 5 18 18 5 9;
27 0 6 1 1 6 0;
3 1 6 26 26 6 1;
8 2 6 22 22 6 2;
13 3 6 18 18 6 3;
18 4 6 14 14 6 4;
23 5 6 10 10 6 5;
28 6 6 6 6 6 6;
4 7 6 2 2 6 7;
9 8 6 27 27 6 8;
14 9 6 23 23 6 9;
17 0 7 6 6 7 0;
22 1 7 2 2 7 1;
27 2 7  27 27 7 2;
3 3 7 23 23 7 3;
8 4 7 19 19 7 4;
13 5 7 15 15 7 5;
18 6 7 11 11 7 6;
23 7 7 7 7 7 7;
28 8 7 3 3 7 8;
4 9 7 28 28 7 9;
7 0 8 11 11 8 0;
```

```
12 1 8 7 7 8 1;
17 2 8 3 3 8 2;
22 3 8 28 28 8 3;
27 4 8 24 24 8 4;
3 5 8 20 20 8 5;
8 6 8 16 16 8 6;
13 7 8 12 12 8 7;
18 8 8 8 8 8 8;
23 9 8 4 4 8 9;
26 0 9 16 16 9 0;
2 1 9 12 12 9 1;
7 2 9 8 8 9 2;
12 3 9 4 4 9 3;
17 4 9 0 0 9 4;
22 5 9 25 25 9 5;
27 6 9 21 21 9 6;
3 7 9 17 17 9 7;
8 8 9 13 13 9 8;
13 9 9 9 9 9 9;
16 0 10 21 21 10 0;
21 1 10 17 17 10 1;
26 2 10 13 13 10 2;
2 3 10 9 9 10 3;
7 4 10 5 5 10 4;
12 5 10 1 1 10 5;
17 6 10 26 26 10 6;
22 7 10 22 22 10 7;
27 8 10 18 18 10 8;
3 9 10 14 14 10 9;
6 0 11 26 26 11 0;
11 1 11 22 22 11 1;
16 2 11 18 18 11 2;
21 3 11 14 14 11 3;
26 4 11 10 10 11 4;
2 5 11 6 6 11 5;
7 6 11 2 2 11 6;
12 7 11 27 27 11 7;
17 8 11 23 23 11 8;
22 9 11 19 19 11 9;
25 0 12 2 2 12 0;
1 1 12 27 27 12 1;
6 2 12 23 23 12 2;
11 3 12 19 19 12 3;
16 4 12 15 15 12 4;
21 5 12 11 11 12 5;
26 6 12 7 7 12 6];

for i=1:4
if i==1
    x11(1:7)=X1(1:7);
end
if i==2
    x11(1:7)=X1(8:14);
end
```

```
 if i==3
    x11(1:7)=X1(15:21);
end

if i==4
    x11(1:7)=X1(22:28);
end

%Comparação das autossequências e identificação do decimal correspondente do
usuário 1.
d=1000;
for k=1:95 %Busca nas autossequências em X que contem as autossequências do
%usuário1.
xtab(1:7)=X(k,1:7);
dcal=norm(xtab-x11);
if dcal<d
    d=dcal;
    X1_F(i,1:7)=xtab;
    S1(i)=10*(xtab(6))+xtab(7)

end
end
end

%Decodificado de canal do usuário 2

%Declaração das palavras códigos do usuário 2.

X=[1 2 3 12 12 3 2;
21 3 3 3 3 3 3;
12 4 3 23 23 3 4;
3 5 3 14 14 3 5;
23 6 3 5 5 3 6;
14 7 3 25 25 3 7;
5 8 3 16 16 3 8;
25 9 3 7 7 3 9;
6 0 4 11 11 4 0;
26 1 4 2 2 4 1;
17 2 4 22 22 4 2;
8 3 4 13 13 4 3;
28 4 4 4 4 4 4;
19 5 4 24 24 4 5;
10 6 4 15 15 4 6;
1 7 4 6 6 4 7;
21 8 4 26 26 4 8;
12 9 4 17 17 4 9;
22 0 5 21 21 5 0;
13 1 5 12 12 5 1;
4 2 5 3 3 5 2;
24 3 5 23 23 5 3;
15 4 5 14 14 5 4;
6 5 5 5 5 5 5 ;
26 6 5 25 25 5 6;
17 7 5 16 16 5 7;
```

```
8 8 5 7 7 5 8;
28 9 5 27 27 5 9;
9 0 6 2 2 6 0;
0 1 6 22 22 6 1;
20 2 6 13 13 6 2;
11 3 6 4 4 6 3;
2 4 6 24 24 6 4;
22 5 6 15 15 6 5;
13 6 6 6 6 6 6;
4 7 6 26 26 6 7;
24 8 6 17 17 6 8;
15 9 6 8 8 6 9;
25 0 7 12 12 7 0;
16 1 7 3 3 7 1;
7 2 7 23 23 7 2;
27 3 7 14 14 7 3;
18 4 7 5 5 7 4;
9 5 7 25 25 7 5;
0 6 7 16 16 7 6;
20 7 7 7 7 7 7;
11 8 7 27 27 7 8;
2 9 7 18 18 7 9;
12 0 8 22 22 8 0;
3 1 8 13 13 8 1;
23 2 8 4 4 8 2;
14 3 8 24 24 8 3;
5 4 8 15 15 8 4;
25 5 8 6 6 8 5;
16 6 8 26 26 8 6;
7 7 8 17 17 8 7;
27 8 8 8 8 8 8;
18 9 8 28 28 8 9;
28 0 9 3 3 9 0;
19 1 9 23 23 9 1;
10 2 9 14 14 9 2;
1 3 9 5 5 9 3;
21 4 9 25 25 9 4;
12 5 9 16 16 9 5;
3 6 9 7 7 9 6;
23 7 9 27 27 9 7;
14 8 9 18 18 9 8;
5 9 9 9 9 9 9;
15 0 10 13 13 10 0;
6 1 10 4 4 10 1;
26 2 10 24 24 10 2;
17 3 10 15 15 10 3;
8 4 10 6 6 10 4;
28 5 10 26 26 10 5;
19 6 10 17 17 10 6;
10 7 10 8 8 10 7;
1 8 10 28 28 10 8;
21 9 10 19 19 10 9;
2 0 11 23 23 11 0;
22 1 11 14 14 11 1;
13 2 11 5 5 11 2;
4 3 11 25 25 11 3;
```

```
24 4 11 16 16 11 4;
15 5 11 7 7 11 5;
6 6 11 27 27 11 6;
26 7 11 18 18 11 7;
17 8 11 9 9 11 8;
8 9 11 0 0 11 9;
18 0 12 4 4 12 0;
9 1 12 24 24 12 1;
0 2 12 15 15 12 2;
20 3 12 6 6 12 3;
11 4 12 26 26 12 4;
2 5 12 17 17 12 5;
22 6 12 8 8 12 6];

for i=1:4
if i==1
    x22(1:7)= X2(1:7);
end
if i==2
    x22(1:7)= X2(8:14);
end
if i==3
    x22(1:7)= X2(15:21);
end
if i==4
    x22(1:7)= X2(22:28);
end

%Comparação das autossequências e identificação do decimal correspondente do
%usuário 2.
d=1000;
for k=1:95 %Busca nas autossequências em X que contem as autossequências do
%usuário2.
xtab(1:7)=X(k,1:7);
dcal=norm(xtab-x22);
if dcal<d
    d=dcal;
    X2_F(i,1:7)=xtab;
    S2(i)=10*(xtab(6))+xtab(7)

end
end
end
end


%Decodificador_ASCII


%Rotina para Decoficar os decimais no correspondente caracter ASCII
%recebido dos usuários.
usuario1 = '    ';
usuario2 = '    ';
m=1:4;
for i=1:2;
    if i==1
```

```
        respusuario=respusuario1;%Lê os 32 decimais do usuário 1(matriz
8x4).
    else
        respusuario=respusuario2;%Lê os 32 decimais do usuário 2(matriz
8x4).
    end

for K=1:8;
    for n=1:4;
rusuario=respusuario(K,n);
switch (rusuario);
case 32;m=' ';
case 33;m='!';
case 34;m='”';
case 35;m='#';
case 36;m='$';
case 37;m='%';
case 38;m='&';
case 39;m='’';
case 40;m='(';
case 41;m=')';
case 42;m='*';
case 43;m='+';
case 44;m=',';
case 45;m='-';
case 46;m='.';
case 47;m=;'/'
case 48;m='0';
case 49;m='1';
case 50;m='2';
case 51;m='3';
case 52;m='4';
case 53;m='5';
case 54;m='6';
case 55;m='7';
case 56;m='8';
case 57;m='9';
case 58;m=':';
case 59;m=';';
case 60;m='<';
case 61;m='=';
case 62;m='>';
case 63;m='?';
case 64;m='@';
case 65;m='A';
case 66;m='B';
case 67;m='C';
case 68;m='D';
case 69;m='E';
case 70;m='F';
case 71;m='G';
case 72;m='H';
case 73;m='I';
case 74;m='J';
case 75;m='K';
```

```
case 76;m='L';
case 77;m='M';
case 78;m='N';
case 79;m='O';
case 80;m='P';
case 81;m='Q';
case 82;m='R';
case 83;m='S';
case 84;m='T';
case 85;m='U';
case 86;m='V';
case 87;m='W';
case 88;m='X';
case 89;m='Y';
case 90;m='Z';
case 91;m='[';
case 92;m='\';
case 93;m=']';
case 94;m='^';
case 95;m='_';
case 96;m='`';
case 97;m='a';
case 98;m='b';
case 99;m='c';
case 100;m='d';
case 101;m='e';
case 102;m='f';
case 103;m='g';
case 104;m='h';
case 105;m='i';
case 106;m='j';
case 107;m='k';
case 108;m='l';
case 109;m='m';
case 110;m='n';
case 111;m='o';
case 112;m='p';
case 113;m='q';
case 114;m='r';
case 115;m='s';
case 116;m='t';
case 117;m='u';
case 118;m='v';
case 119;m='w';
case 120;m='x';
case 121;m='y';
case 122;m='z';
case 123;m='{';
case 124;m='|';
case 125;m='}';
case 126;m='~';
    otherwise
        m=' ';
end
m(K,n)=m;
if i==1
```

```
usuario1(K,n)=m(K,n);
else
usuario2(K,n)=m(K,n);
end
    end
end
end

saidausuario1=[usuario1(1,1:4) usuario1(2,1:4) usuario1(3,1:4)
usuario1(4,1:4) usuario1(5,1:4) usuario1(6,1:4) usuario1(7,1:4)
usuario1(8,1:4)]% Informação Decodificada do usuário 1.
saidausuario2=[usuario2(1,1:4) usuario2(2,1:4) usuario2(3,1:4)
usuario2(4,1:4) usuario2(5,1:4) usuario2(6,1:4) usuario2(7,1:4)
usuario2(8,1:4)] % Informação Decodificada do usuário 2.
```

REFERÊNCIAS

[1] R. E. Blahut,“Transform techniques for error-control codes”, *IBM J. Res. Dev*. v. 23, p. 299-315, Maio 1979.

[2] I. S. Reed & T.K. Truong,“The use of finite fields to compute convolutions”, *IEEE Trans. on Information Theory*, vol. 21, n. 2, p. 208-213, Março 1975.

[2] T. Toivonen & J. Heikkilã,“Video filtering with Fermat number theoretical transforms using redidue number system”, *IEEE Trans. Circuits Syst. Video tech.,* v. 16, n. 1, p. 92-101, Janeiro 2006.

[4] J.M. Pollard, “The Fast Fourier Transform in a Finite Field”, *Mathematics of Computation*, v. 25, No. 114, p. 365-374, April 1971.

[5] R.M. Campello de Souza, H. M. de Oliveira, and A. N. Kauffman, “Trigonometry in finite fields and a new Hartley Transform” In: *Proc. IEEE Int. Symp. Information Theory*, Boston, MA, 1998, p. 293.

[6] R.M. Campello de Souza, H. M. de Oliveira, and A. N. Kauffman, “Efficient multiplex for band-limited channels: Galois-field multiple access”, In: *Proc. of the Workshop on Coding and Criptography*, Cambridge, United Kingdom, 1999, p. 235-241.

[7] R. M. Campello de Souza and H. M. de Oliveira,“The complex Hartley transform over a finite field, P. G. Farnell, M. Darnell, and B. Honary, Eds.”, In: *Coding, Communications and Broadcasting*, 1ª ed. Hertfordshire: Research Studies Press, Jonh Wiley, 2000, p. 267-276.

[8] H. M. de Oliveira and R. M. Campello de Souza,“Orthogonal multilevel spreading sequence design, P. G. Farnell, M. Darnell, and B. Honary, Eds.”, In: *Coding, Communications and Broadcasting*, 1ª ed. Hertfordshire: Research Studies Press, Jonh Wiley, 2000, p. 291-303.

[9] R. M. Campello de Souza, H. M. de Oliveira, L.B. Espínola e M. M. Campello de Souza, "Transformadas Numéricas de Hartley", In: *Anais do XVIII Simpósio Brasileiro de Telecomunicações*, pp. 357 - 366, Gramado, RS, setembro 2000.

[10] D. Silva, R. M. Campello de Souza, H. M. de Oliveira, L. B. E. Palma and M.M.Campello de Souza, "A Transformada Numérica de Hartley e Grupos de Inteiros Gaussianos", *Revista da Sociedade Brasileira de Telecomunicações*, Campinas, SP, v. 17, n.1, pp. 48-57, 2002.

[11] M. A. Suhail and M. S. Obaidat, "Digital watermarking-based DCT and JPEG model", In: *IEEE Trans.on Instrumentation and Measurement*, v.52, n. 5, p. 1640-1647, Outubro 2003.

[12] H. Park, Y. Park, and S. Oh, "L/M-fold image resizing in block-DCT domain using symmetric convolution", In: *IEEE Trans.on Image Processing*, v.12, n. 9, p. 1016-1034, September 2003.

[13] H. Park, Y. Park, "Design and analysis of an image resizing filter in the block-dct domain", In: *IEEE Trans.on Circuits and Systems for Video Technology*, v.14, n. 2, p.1016-1034, February 2004.

[14] S.-C. Pei and J.-J. Ding, "Generalized eigenvectors and fractionalization of offset DFTs and DCTs", *IEEE Transactions on Signal Processing*, v.52, n.17, p.1661-1680, Julho 2004.

[15] M. M. Campello de Souza, H. M. de Oliveira, R. M. Campello de Souza and M. M. Vasconcelos, "The Discrete Cosine Transform over Prime Finite Fields", In: *Proceedings of the 11th International Conference on Telecommunications*, ser. Lecture Notes in Computer Science, J. N. de Souza, P. Dini, and P. Lorentz, Eds. Berlin: Springer, 2004, p. 482-487.

[16] R. M. C. de Souza, H. M. de Oliveira, M. M. Campello de Souza e M. M. Vasconcelos, "A Transformada Discreta do Seno em um Corpo Finito", In: *Anais do XXVIII Congresso Nacional de Matemática Aplicada e Computacional*, Sao Paulo, Brasil, 2005.

[17] J. B. Lima, *Trigonometria sobre Corpos Finitos: Novas Definições e Cenários de Aplicação*. Recife, 2008. Tese (Doutorado em Engenharia Eletrica) - DES, Universidade Federal de Pernambuco.

[18] J. B. Lima and R. M. Campello de Souza, "New Trigonometric Transforms over Prime Finite Fields for Image Filtering", In: *Proceedings of the International Telecommunications* , Fortaleza, Brasil, 2006.

[19] J. B. Lima and R. M. C. de Souza, "Uma Marca D'água Digital Baseada na Transformada do Cosseno sobre Corpos Finitos", In: *Anais do XXII Simpósio Brasileiro de Telecomunicações*, Campinas, Brasil, 2005.

[20] J. B. Lima, R. M. Campello de Souza and D. Panario, "Blind Sequence Separation Based on the Eigenstructure of Finite Field Transforms", In: *Anais do XXVI Simpósio Brasileiro de Telecomunicações*, Rio de Janeiro, Brasil, 2008.

[21] S. C. E, "A Mathematical Theory of Communication", *Bell. Syst. Tec. Jour.*, v. 27 pp. 374 - 423, 623 - 656, July, 1948.

[22] S. C. E, "Two-way Communication Channels", In: *Proceedings 4th Berkeley Symp. Math. Stat. Prob.*, v. 1, pp. 611-644, 1961.

[23] H. A. Cabral, *Codificação para Canal de Acesso Múltiplo Síncrono*. Recife, 1994. Dissertação (Mestrado em Engenharia Eletrica) - DES, Universidade Federal de Pernambuco.

[24] M. L. M. G. Alcoforado, *Implementação Algoritmica de Códigos Lineares para o Canal Aditivo com Dois Usuários Binários*, Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica, Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife, 1999.

[25] P. Z. Fan, M. Darnell, and B. Honary “Superimposed Codes for thr Multiacess Binary Adder Channel”. *IEEE Trans. Inform. Theory*, v. 41, n. 4, pp.1178-1182, 1995.

[26] H. Liao, *Multiple Access Channels.* USA, 1972. Tese (Doutorado) – Dept. Elec. Eng., University of Hawaii.

[27] T. Kasami, and S. Lin, “Coding for a Multiple-Access Channel”. *IEEE Trans. Inform. Theory*, v. IT-22, n. 2, pp.129-137, 1976.

[28] V.C. da Rocha Jr. and J.L. Massey, “A New Approach to the Design of Codes for the Binary Adder Channel”, In: M. Ganley, Ed., *IMA Conf. Series Cryptography and Coding III*, pp. 179-185, Claredon Press, Oxford, 1993.

[29] S. C. Chang and E. J. Weldon Jr., “Coding for T-user Multiple-Access Channels”. *IEEE Trans. Inform. Theory,* v. IT-25, pp.684-691, November 1979.

[30] W. H. Kautz and R.C. Singleton,“Nonrandom binary superimposed codes”. *IEEE Trans. Inform. Theory*, v. IT-10, n. 4, pp.363-377, 1964.

[31] R. T. Chien and W. D. Frazer,“No application of coding theory to document retrieval”. *IEEE Trans. Inform. Theory*, v. IT-12, n. 2, pp.92-96, 1966.

[32] R. M. Campello de Souza e H. M. de Oliveira, “Eigensequences for Multiuser Communication over the Real Adder Channel”, *VI International Telecommunications Sympo*sium (ITS2006), September 3-6, Fortaleza-CE, Brazil.

[33] J. H. McClellan and T. W. Parks, “Eigenvalue and Eigenvector Decomposition of the Discrete Fourier Transform”, *IEEE Transactions on Audio and Electroacoustics,* vol. AU-20, pp. 66-74, 1972.

[34] G. Cariolaro, T. Erseghe and P. Kraniauskas,"The Fractional Discrete Cosine Transform", *IEEE Transactions on Signal Processing*, v. 50, p.902-911, Abril 2002.

[35] S.-C. Pei and M.-H. Yeh, "The Discrete Fractional Cosine and Sine Transforms", *IEEE Transactions on Signal Processing*, v. 49, p. 1198-1207, Junho 2001.

[36] C. C. Tseng, "Eigenvalues and Eigenvectors of Generalized DFT, Generalized DHT and DST - IV Matrices", *IEEE Trans. on Signal Processing*, vol. SP-50, No. 4, pp. 866-877, April 2002.

[37] J. B. Lima and R. M. Campello de Souza, "Uma técnica de múltiploacesso baseado na auto-estrutura das transformadas trigonométricas," in *Anais do XXIII Simpósio Brasileiro de Telecomunicações*, Recife, Brasil, 2007.

[38] H. M. Cavalcanti and R. M. Campello de Souza, "Autossequências da Transformada Numérica de Fourier," in *Anais do XXIII Simpósio Brasileiro de Telecomunicações*, Recife, Brasil, 2007.

[39] J. B. Lima, R. M. Campello de Souza and D. Panario, "The Eigenstructure of Finite Field Trigonometric Transforms", *Linear Algebra and its applications*, No. 435, pp. 1956-1971, April 2011.

[40] J. B. Lima e R. M. Campello de Souza, "Comunicação Multiusuário Baseada na Transformada Discreta Fracional de Fourier". In *Anais do XXIX Simpósio Brasileiro de Telecomunicações, SBrT 2011*, 2 a 5 de outubro de 2011, Curitiba.

[41] V. Ipatov, *Spread Spectrum and CDMA Principles and Applications*, John Wiley, 2005.

[42] R. M. Campello de Souza, "Transformadas em Corpos Finitos para Codificação de Canal", *Revista da Sociedade Brasileira de Telecomunicações*, N. 1, vol. 5, p. 41-57, 1990.

[43] R. M. Campello de Souza, E. S. V. Freire and H. M. de Oliveira, “Fourier Codes”, In: *Proceedings of the Tenth International Symposium on Communication Theory and Applications, ISCTA’09*, pp. 370-375, Ambleside, Lake District, UK, 2009.

[44] E. S. V. Freire, *Construção de Códigos de Bloco Lineares via Transformadas Digitais*.. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica, Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife, 2009.

[45] R. E. Blahut, *Fast Algorithms for Signal Processing*, Cambridge University Press, 2010.

[46] Chris K. Caldwell, 2011. Disponível em http://primes.utm.edu/largest.html. Acesso em 08/08/2001.

[47] S. Lin and D. J. Costello, Jr., *Error Control Coding*, Pearson Prentice Hall, 2004.

[48] D. M. Burton, *Elementary Number Theory*, 7ª. Ed.,McGraw Hill, 2005.

[49] D. F. Elliott and K. R. Rao, *Fast Transforms – Algorithms, Analyses, Applications*, Academic Press, 1982.

[50] I. N. Herstein, *Topics in Algebra*, John Wiley, 1975.

[51] S. C. Chan e K. L. Ho, “Direct Method for Computing Sinusoidal Transforms”, *IEEE Proceedings*, vol. 137, Pt. F, No. 6, pp. 433–442, 1990.

[52] M. Jeruchim, C. Balaban and P. Shanmugan, *Simulation of Communications Systems*: *Modeling, Methodology and Techniques*, 2ª. Ed. , New York, Kluwer Academic Publishers, 2000.

[53] M. C. Jeruchim, “Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems”, *IEEE Journal on Selected Areas in Communications*, Vol. Sac-2, No. 1, pg.153 – 170, Janeiro 1984.

[54] J. B. Lima and R. M. Campello de Souza, “The Finite Field Fractional Fourier Transform”, *International Conference on Acoustics, Speech and Signal Processing*, Dallas, March 2010.

[55] J.G. Proakis, *Digital Communications*, 4a. Ed., McGraw Hill, 2000.