IMPLEMENTAÇÃO DE FILTROS DE WIENER COM POSTO E PRECISÃO REDUZIDOS

Bruno de Carvalho Costa

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Marcello Luiz Rodrigues de Campos, Ph.D.

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. José Antonio Apolinário Junior, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2006

COSTA, BRUNO DE CARVALHO

Implementação de Filtros de Wiener com posto e precisão reduzidos [Rio de Janeiro] 2006

XII, 108 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2006)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1.Processamento de Sinais 2.CORDIC 3.SVD 4.Filtro de Wiener 5. Redução de posto

I.COPPE/UFRJ II.Título (série)

Agradecimentos

Agradeço aos meus pais Belchior e Sueli, ao meu irmão Gustavo, à minha madrinha Flávia e aos meus avós Maria de Lourdes, Syrio, Terezinha e Alberto pela compreensão, carinho e torcida. Devo tudo que sou à minha família, que sempre foi referência e ponto de equilíbrio para todos os momentos da minha vida.

À minha futura esposa Ana Lúcia pela paciência e apoio incondicional desde o dia em que optei seguir o caminho de um mestre. Nosso amor superou muitas barreiras em prol desta dissertação.

Gostaria de agradecer ao meu orientador e amigo Marcello Campos por acreditar sempre no resultado deste trabalho. Sua dedicação e vontade me fizeram vencer as dificuldades encontradas e me motivaram ao longo do curso de mestrado para que o objetivo final fosse alcançado.

Gostaria de agradecer aos professores e funcionários do LPS e COPPE/UFRJ, especialmente ao Prof. Gélson Mendonça, Prof. Eduardo Silva, Prof. Paulo Diniz, Prof. Sérgio Lima Netto, Prof. José Seixas e Prof. Mariane Petraglia que me guiaram com a maestria necessária para que eu me formasse como homem, engenheiro e como mestre.

Agradeço aos colegas de mestrado Tadeu Nagashima, Michel Tcheou, Filipe Diniz, Leonardo Baltar e William Roger, sem os quais esta pós-graduação não teria sido a mesma. Vocês foram mais que amigos, sendo eternizados em minha memória.

Aos meus amigos e colegas da Embratel mais que "especiais" Márcio Tóros, Marco Aurélio Borda, Alexandre Soares, Roberto Maia, Carlos Jardim, Adda Pimentel, Mauro Morais, Sérgio Cunha, Marta Gonçalves, Maria Elisa, Luis Soeiro e Henrique Cézar que acompanharam de perto cada passo e facilitaram muito o dia a dia nestes três anos de convivência na Embratel. Agradeço também aos demais colegas de Projetos Especiais por sermos uma família.

Agradeço ao CNPq pelos três meses de bolsa no desenvolvimento deste trabalho.

Por fim agradeço aos meus amigos Fabiana Costa, Márcio Araújo e a todos que contribuíram direta ou indiretamente para a conclusão deste trabalho.

Muito obrigado a todos vocês.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Bruno de Carvalho Costa

Março/2006

Orientador: Marcello Luiz Rodrigues de Campos

Programa: Engenharia Elétrica

A história das Telecomunicações é marcada por fortes avanços tecnológicos, traduzidos em décadas de pesquisas, desenvolvimento de algoritmos mais eficientes e implementações em hardware cada vez mais rápidos. Uma das técnicas mais conhecidas para reduzir a complexidade de um algoritmo é reduzir o posto de matrizes de ordem elevada, concentrando a informação destas matrizes em alguns índices pré-determinados e aumentando a sua eficiência como um todo. A SVD (*Decomposição em Valores Singulares*), por sua característica peculiar de concentrar a informação da matriz nos valores singulares, é uma das ferramentas existentes para reduzir o posto. Nesta tese propomos métodos de obter a SVD com precisão variável (CORDIC) durante e após a redução de posto, e comparamos os resultados obtidos em uma aplicação de estimação.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Bruno de Carvalho Costa

March/2006

Advisor: Marcello Luiz Rodrigues de Campos

Department: Electrical Engineering

The Telecommunications history is marked by great technological advances, translated in decades of research, development of more efficient algorithms and faster hardware implementation. One of the most well known techniques to reduce algorithm complexity is to reduce the rank of high order matrix, concentrating the information in some indexes and increasing the algorithms efficiency. SVD (Singular Value Decomposition) with its peculiar characteristic of concentrating matrix information's in singular values, is one of the available tools to do rank reduction. This work proposes methods to perform finite-precision singular value decomposition (CORDIC's algorithms) during and after rank reduction, and compare the results in an estimation application.

Sumário

1 Introdução **1**

1.1 Motivação 1

1.2 Organização da Tese 2

2 Algoritmos CORDIC **4**

2.1 Introdução 4

2.1.1 Dimensão das Rotações 5

2.2 Algoritmo CORDIC 5

2.2.1 Sistemas de Coordenadas 8

2.3 Modos de Operação 12

2.3.1 Modo de Rotação 12

2.3.2 Modo de Vetorização 14

2.4 Convergência do Algoritmo 15

2.4.1 Seqüências de Deslocamento 16

2.5 Funções CORDIC 18

2.6 Fator de Correção de Escala 21

2.7 Acurácia do Algoritmo 22

2.8 Implementação: CORDIC no Matlab 23

2.8.1 Função Tangente Cordic (tan_c) 24

2.8.2 Função Tangente Inversa Cordic ($atan\_c$) 26

2.8.3 Função Cosseno Cordic (cos_c) 28

2.8.4 Função Seno Cordic (sin_c) 29

2.8.5 Função Secante Cordic (sec_c) 30

2.8.6 Função Cossecante Cordic (csc_c) 31

2.8.7 Função Soma Cordic ($soma\_c$) 32

2.8.8 Função Multiplicação Cordic ($mult\_c$) . . . 33
2.8.9 Função Divisão Cordic (div_c) . . . 35
2.8.10 Função Tangente Hiperbólica Cordic ($tanh\_c$) . . . 36
2.8.11 Função Tangente Inversa Hiperbólica Cordic ($atanh\_c$) . . . 38
2.8.12 Função Cosseno Hiperbólico Cordic ($cosh\_c$) . . . 39
2.8.13 Função Seno Hiperbólico Cordic ($sinh\_c$) . . . 41
2.8.14 Função Raiz Quadrada Cordic ($sqrt\_c$) . . . 41
2.8.15 Função Exponencial Cordic (exp_c) . . . 42
2.8.16 Função Logaritmo Neperiano Cordic (ln_c) . . . 44
2.9 Resultados . . . 46

3 Decomposição em Valores Singulares 47
3.1 Introdução . . . 47
3.2 Conceito de SVD . . . 47
3.3 Métodos de Jacobi . . . 49
3.3.1 Decomposição de Schur . . . 51
3.3.2 Método Clássico de Jacobi . . . 52
3.3.3 Algoritmos Cyclic-by-Row e Cyclic-by-Column . . . 52
3.4 Implementação: SVD pelo Método de Jacobi no Matlab . . . 53
3.4.1 Exemplo com matriz 3×3 . . . 54
3.4.2 Exemplo com matriz 5×5 . . . 55
3.4.3 Resultados . . . 56

4 SVD com precisão CORDIC 58
4.1 Implementação: Algoritmo SVD_CORDIC no Matlab . . . 58
4.1.1 Região de Convergência da SVD_CORDIC . . . 62
4.2 Testes com Matrizes Simétricas 2×2 . . . 63
4.2.1 SVD com precisão de 3 bits . . . 65
4.2.2 SVD com precisão de 5 bits . . . 65
4.2.3 SVD com precisão de 7 bits . . . 65
4.2.4 SVD com precisão de 10 bits . . . 66
4.2.5 SVD com precisão de 15 bits . . . 66
4.3 SVD_CORDIC com posto reduzido . . . 66

4.3.1 Redução do posto após a quantização CORDIC 66
4.3.2 Redução do posto em conjunto com a quantização CORDIC . 69
4.4 Resultados da SVD_CORDIC 72

5 Filtro de Wiener com Posto e Precisão reduzidos 73
5.1 Introdução 73
5.2 Filtros de Wiener 73
5.3 Variância do erro e Coerência 75
5.4 Coordenadas padrão 77
5.5 Redução de Posto 79
5.6 Implementação: Filtro de Wiener e redução de posto com precisão variável no Matlab 80
5.6.1 Wiener com precisão do Matlab e posto completo 81
5.6.2 Wiener com precisão do Matlab e posto reduzido 83
5.6.3 Wiener com precisão variável pela *SVD_CORDIC* e posto completo 86
5.6.4 Wiener com precisão variável pela *SVD_CORDIC* e posto reduzido "*a posteriori*" 89
5.6.5 Wiener com precisão variável pela *SVD_CORDICRED* e posto reduzido em conjunto 96
5.7 Resultados da aplicação 101

6 Conclusão 103
6.1 Contribuições da Tese 103
6.2 Propostas de Trabalhos Futuros 106

Referências Bibliográficas 107

Lista de Figuras

2.1 Rotação de um vetor v por um ângulo α 6
2.2 Sistema de Coordenadas Circular 10
2.3 Sistema de Coordenadas Linear 10
2.4 Sistema de Coordenadas Hiperbólico 11
2.5 Trajetória de uma transformação CORDIC no modo de rotação com sistema de coordenadas circular. 13
2.6 Trajetória de uma transformação CORDIC no modo Y-Reduction com sistema de coordenadas circular. 15
2.7 Trajetória das rotações CORDIC no cálculo do seno de 68°. 20
2.8 Modelo dos gráficos 24
2.9 Tangente Cordic 25
2.10 Gráfico da função Tangente Cordic 26
2.11 Tangente Inversa Cordic 27
2.12 Gráfico da função Tangente Inversa Cordic 27
2.13 Cosseno Cordic 28
2.14 Gráfico da função Cosseno Cordic 29
2.15 Seno Cordic 30
2.16 Secante Cordic 30
2.17 Gráfico da função Secante Cordic 31
2.18 Cossecante Cordic 32
2.19 Soma Cordic 33
2.20 Gráfico da função Soma Cordic 33
2.21 Multiplicação Cordic 34
2.22 Gráfico da função Multiplicação Cordic 35
2.23 Divisão Cordic 36

2.24 Gráfico da função Divisão Cordic 36
2.25 Tangente Hiperbólica Cordic 37
2.26 Gráfico da função Tangente Hiperbólica Cordic 38
2.27 Tangente Inversa Hiperbólica Cordic 39
2.28 Gráfico da função Tangente Inversa Hiperbólica Cordic 39
2.29 Cosseno Hiperbólico Cordic 40
2.30 Gráfico da função Cosseno Hiperbólico Cordic 40
2.31 Seno Hiperbólico Cordic 41
2.32 Raiz Quadrada Cordic 42
2.33 Gráfico da função Raiz Quadrada Cordic 42
2.34 Exponencial Cordic 43
2.35 Gráfico da função Exponencial Cordic 44
2.36 Logaritmo Neperiano Cordic 45
2.37 Gráfico da função Logaritmo Neperiano Cordic 45

4.1 Precisão dos valores singulares variando de 1 a 20 bits 64
4.2 Redução do posto após a quantização CORDIC 67
4.3 Redução do posto em conjunto com a quantização CORDIC 70

5.1 Exemplo de Estimação Linear 78
5.2 Filtro de Wiener no sistema de coordenadas padrão 78
5.3 Norma do erro acumulado pela quantização do Matlab 82
5.4 Valores singulares de $\mathbf{R}_{yy}$ 83
5.5 Figura referente a Posto 40 84
5.6 Figura referente a Posto 10 84
5.7 MSE $\times$ Posto de $\mathbf{R}_{yy}$ 85
5.8 Wiener com 10 bits de precisão 87
5.9 Wiener com 7 bits de precisão 87
5.10 Wiener com 5 bits de precisão 88
5.11 Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 10 bits 90
5.12 Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 10 bits 90

5.13 Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 8 bits 92
5.14 Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 8 bits 92
5.15 Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 5 bits 94
5.16 Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 5 bits 95
5.17 Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 10 bits e posto 50 97
5.18 Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 10 bits e posto 30 98
5.19 Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 8 bits e posto 50 . 99
5.20 Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 8 bits e posto 30 . 100

Lista de Tabelas

2.1 Valores dos ângulos fixos para a seqüência dos números naturais . . . 7
2.2 Sistema de Coordenadas no CORDIC 9
2.3 Seqüência de Deslocamento do CORDIC 17
2.4 Seqüência de Deslocamento modificada do CORDIC 18
2.5 Funções calculadas pelo CORDIC 19
2.6 Exemplo das iterações CORDIC para o cálculo do seno de 68° 20

4.1 Número de operações aritméticas, trigonométricas e funções do Matlab que compõem cada função da SVD_J 59
4.2 Número de operações aritméticas, trigonométricas e funções do Matlab que compõem cada função da SVD_C 62

5.1 Norma do erro e erro médio quadrático com precisão do Matlab . . . 82
5.2 Erro médio quadrático com precisão do Matlab para diferentes postos reduzidos 86
5.3 Erro médio quadrático com precisão variável dada pelo CORDIC . . 89
5.4 Erro médio quadrático com precisão CORDIC de 10 bits e precisão do Matlab para diferentes postos reduzidos 91
5.5 Erro médio quadrático com precisão CORDIC de 8 bits e precisão do Matlab para diferentes postos reduzidos 93
5.6 Erro médio quadrático com precisão CORDIC de 5 bits e precisão do Matlab para diferentes postos reduzidos 96
5.7 Erro médio quadrático com precisão CORDIC de 10 bits e precisão do Matlab para diferentes postos reduzidos 98
5.8 Erro médio quadrático com precisão CORDIC de 8 bits e precisão do Matlab para diferentes postos reduzidos. 101

Capítulo 1

Introdução

1.1 Motivação

A história das Telecomunicações é marcada por fortes avanços tecnológicos, traduzidos em décadas de pesquisas, desenvolvimento de algoritmos mais eficientes e implementações em hardware cada vez mais rápidos.

Quando falamos de eficiência de um algoritmo nos referimos ao seu tempo de execução computacional, cuja grandeza depende de alguns parâmetros, como o tipo de hardware e o número de entradas do software. Por este motivo, medimos o tempo computacional por métodos analíticos, de forma a determinar a complexidade de um algoritmo independente da linguagem de programação, compilador ou hardware usados.

Citando o caso específico da telefonia celular, em que códigos pseudo-aleatórios e multiplicações de matrizes de ordem elevada são constantemente usados à medida que as gerações vão sendo desenvolvidas (por exemplo, a terceira e a quarta geração de celulares), uma das técnicas mais simples e conhecidas para diminuir a complexidade desses algoritmos é reduzir o posto das matrizes. Para isto, é necessário concentrar a informação destas matrizes em alguns índices pré-determinados, para que a redução de posto seja mais eficiente.

A SVD (*Decomposição em Valores Singulares*) é uma das ferramentas existentes para redução do posto, por sua característica peculiar de concentrar a informação da matriz nos valores singulares.

Apesar de existirem diversos algoritmos para calcular a SVD, tais como os

métodos de Golub-Kahan SVD, o Golub-Reinsch SVD, R-SVD, e o GSVD [1], os métodos de Jacobi são os mais usados por sua característica de computação paralela.

Nesta tese, é implementada a SVD pelo método de Jacobi e são propostos dois métodos de obter a SVD com precisão variável (CORDIC) durante e após a redução de posto, para comparação dos resultados obtidos em uma aplicação de estimação.

O CORDIC (COordinate Rotation DIgital Computer) é um algoritmo rápido e simples, possuindo características geométricas que permitem, através do uso de somadores e registradores de deslocamento, realizar cálculos de operações aritméticas complexas. O CORDIC calcula ainda com a precisão de n bits senos, cossenos, tangentes e muitas outras operações, podendo ser utilizado como unidade aritmética de cálculo em qualquer algoritmo.

1.2 Organização da Tese

No Capítulo 2, descrevemos detalhadamente o funcionamento do algoritmo CORDIC, assim como seus modos de operação e suas regiões de convergência. Foram implementadas ao todo 16 funções CORDIC no Matlab, com possibilidade de escolha dos bits de precisão. Para cada função plotamos gráficos e analisamos a região de convergência.

No Capítulo 3, mostramos a SVD, suas características e detalhamos o método de Jacobi para matrizes simétricas $n \times n$. Foi desenvolvido no Matlab a função SVD_JACOBI, que calcula a SVD pelo método clássico de Jacobi.

Os métodos propostos para cálculo da SVD com precisão variável serão mostrados no Capítulo 4. Foi desenvolvida no Matlab as funções SVD_CORDIC e SVD_CORDICRED, juntando os conceitos vistos nos dois capítulos anteriores. Os resultados obtidos com esta implementação podem ser vistos no final deste capítulo, com operações de matrizes 2×2 e 3×3 com precisão de 3, 5, 7, 10 e 15 bits.

No Capítulo 5 detalhamos o conceito da decomposição do filtro de Wiener nas coordenadas padrão, para aplicação da redução de posto. Foram desenvolvidos no Matlab cinco cenários de estimação, com precisão variável e posto reduzidos.

Por fim, no Capítulo 6 apresentamos as conclusões de cada simulação e pro-

pomos uma linha de trabalho futuro na área de redução de posto com precisão variável.

Capítulo 2

Algoritmos CORDIC

2.1 Introdução

Os algoritmos CORDIC (***CO****ordinate* ***R****otation* ***DI****gital* ***C****omputer*) foram desenvolvidos no intuito de melhorar o desempenho de determinadas operações aritméticas complexas, possibilitando implementações mais eficientes de arquiteturas e algoritmos DSP (*Digital Signal Processing*).

Foi com esta motivação que Volder [2] introduziu em 1959 uma técnica de rotações de vetores com operações de soma e deslocamento, possibilitando o cálculo iterativo de funções trigonométricas e funções hiperbólicas. A cada iteração, a precisão do algoritmo é incrementada por um bit de acurácia, o que dá ao algoritmo uma característica peculiar e expande o seu leque de atuação na área de processamento de sinais e sistemas digitais.

O CORDIC é um conjunto de rotações de ângulos fixos calculados por uma seqüência de deslocamento, com direção de rotação determinada por um esquema de controle que garante a convergência do algoritmo. Os ângulos de rotação vão diminuindo a cada iteração, e dependendo do esquema de controle usado, podemos ter diferentes modos de operações possibilitando a implementação de diferentes funções.

Os resultados obtidos por Volder foram aperfeiçoados por outros pesquisadores, destacando-se o trabalho de Walther [3] que generalizou as equações originais para funções hiperbólicas, Meggitt [4] nas computações com base decimal e Daggett [5] na conversão de decimal-binário. Muitos avanços em VLSI (*Very Large-Scale Integration*) só foram possíveis devido ao uso do CORDIC em suas arquiteturas.

Suas rotações, também chamadas de microrrotações, oferecem uma alternativa elegante para o cálculo de funções aritméticas, trigonométricas, hiperbólicas, sistemas de equações lineares, cálculo de autovalores, autovetores, valores singulares e decomposição QR.

2.1.1 Dimensão das Rotações

Originalmente, o CORDIC descrito por Volder lida com rotações no espaço Euclidiano ou Pseudo-Euclidiano, sendo eficiente no que se propôs. Porém, quando é preciso rotacionar vetores em espaços com dimensões maiores que 2, o algoritmo necessariamente decompõe estas rotações em *2D* para depois recompor o produto final, onerando o desempenho do processador. À medida que o tamanho do espaço vai aumentando, torna-se complicado manter a velocidade de implementação por mais rápida que seja a arquitetura do DSP.

Com este raciocínio, Hsiao e Delosme [6] implementaram uma generalização do algoritmo CORDIC permitindo que rotações de vetores de dimensão n sejam decompostas em rotações com dimensões maiores que 2, como 3 e 5, por exemplo, preservando a sua norma original. Esta generalização, conhecida como *CORDIC Householder*, não será abordada nesta tese, e todas as implementações CORDIC realizadas neste trabalho são rotações de vetores bidimensionais.

2.2 Algoritmo CORDIC

Volder desenvolveu o algoritmo CORDIC baseado na forma geral da matriz de rotação de Givens, que rotaciona um vetor $V = (x, y)$ por um ângulo α no plano Euclidiano:

$$\begin{cases} x' = x\cos(\alpha) - y\,\mathrm{sen}(\alpha) \\ y' = y\cos(\alpha) + x\,\mathrm{sen}(\alpha) \end{cases} \tag{2.1}$$

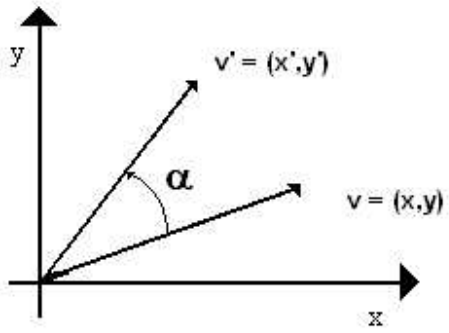

Figura 2.1: Rotação de um vetor v por um ângulo α

Organizando a equação acima, temos:

$$\begin{cases} x' = \cos(\alpha) \cdot [x - y \tan(\alpha)] \\ y' = \cos(\alpha) \cdot [y + x \tan(\alpha)] \end{cases} \tag{2.2}$$

Se os ângulos de rotação forem limitados a $tan(\alpha) = \pm 2^{-i}$, a multiplicação do termo tangente é reduzida a uma operação simples de deslocamento [7]. A direção da rotação pode ser tanto no sentido horário quanto no sentido anti-horário, pois $cos(\alpha) = cos(-\alpha)$. A equação 2.2 pode ser expressa como:

$$\begin{cases} x_{i+1} = K_i[x_i - y_i \cdot \mu_i \cdot 2^{-i}] \\ y_{i+1} = K_i[y_i + x_i \cdot \mu_i \cdot 2^{-i}] \end{cases} \tag{2.3}$$

em que:

$$K_i = \cos(\tan^{-1}(2^{-i})) = \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{2.4}$$

$$\mu_i = \pm 1 \tag{2.5}$$

Se a variável K_i for removida das equações iterativas acima, pode-se rotacionar vetores somente com operações de soma e de deslocamento. A remoção desta variável, chamada de fator de correção de escala, pode ocorrer através de um produto realizado após cada iteração ou pode ocorrer ao final de todas as iterações, como um processo de ganho do algoritmo.

Os ângulos de cada rotação são determinados pelo valor de arcotangentes, que variam de acordo com uma seqüência não decrescente. A tabela abaixo exemplifica passo a passo os ângulos de rotação fixos para a seqüência dos números naturais:

Tabela 2.1: Valores dos ângulos fixos para a seqüência dos números naturais

$\mathbf{tan}(\alpha_i)$	α_i	$\mathbf{cos}(\alpha_i)$
$\tan(\alpha_1) = \frac{1}{1}$	$\alpha_1 = 45°$	$\cos(\alpha_1) = 0.7071$
$\tan(\alpha_2) = \frac{1}{2}$	$\alpha_2 = 26.5650°$	$\cos(\alpha_2) = 0.8944$
$\tan(\alpha_3) = \frac{1}{4}$	$\alpha_3 = 14.0362°$	$\cos(\alpha_3) = 0.9701$
$\tan(\alpha_4) = \frac{1}{8}$	$\alpha_4 = 7.1250°$	$\cos(\alpha_4) = 0.9922$
$\tan(\alpha_5) = \frac{1}{16}$	$\alpha_5 = 3.5763°$	$\cos(\alpha_5) = 0.9980$
$\tan(\alpha_6) = \frac{1}{32}$	$\alpha_6 = 1.7899°$	$\cos(\alpha_6) = 0.9995$
$\ldots$	$\ldots$	$\ldots$

Uma terceira variável z é necessária para acumular os ângulos rotacionados, como exposto na equação a seguir:

$$z_{i+1} = z_i - \mu_i \cdot \tan^{-1}(2^{-i}) \tag{2.6}$$

Por exemplo, se substituirmos os valores de $tan(\alpha_i)$ e $cos(\alpha_i)$ na equação 2.2, e iniciarmos as variáveis x_0, y_0 e z_0 com os valores $\frac{1}{K_i}$, 0 e um ângulo qualquer θ respectivamente, temos ao final de i iterações o cosseno de θ armazenado na variável x_i. Se aplicarmos o método uma vez, temos a precisão de um bit, e assim por diante.

Juntando os conceitos das equações 2.1, 2.2 e 2.3, Volder descreve uma iteração CORDIC composta exclusivamente com operações de soma/subtração e deslocamento da seguinte maneira:

$$\begin{cases} x_{i+1} = x_i - m \cdot \mu_i \cdot y_i \cdot \delta_{m,i} \\ y_{i+1} = y_i + \mu_i \cdot x_i \cdot \delta_{m,i} \\ z_{i+1} = z_i - \mu_i \cdot \alpha_{m,i} \end{cases} \tag{2.7}$$

A cada iteração um vetor $v_i = (x_i, y_i)^T$ será transformado linearmente num vetor $v_{i+1} = (x_{i+1}, y_{i+1})^T$, e a variável z assume o somatório dos ângulos de rotação $\alpha_{m,i}$. A variável m pertencente ao conjunto {1,0,-1} determina o sistema de coordenadas, enquanto a variável μ_i, podendo assumir os valores {1,-1}, irá determinar a direção da rotação, desempenhando um papel fundamental para a convergência do algoritmo. A variável $\delta_{m,i}$ é definida como $\delta_{m,i} = d^{-s_{m,i}}$, sendo d a base do sistema numérico e $s_{m,i}$ uma seqüência de deslocamento de números inteiros, geralmente não decrescente.

Para um melhor entendimento da equação (2.7), vamos estudar separadamente cada uma das variáveis acima com intuito de entender de forma clara o seu propósito dentro do algoritmo. A base binária, usada principalmente em operações computacionais, foi a base escolhida para implementação, sendo portanto $\delta_{m,i} = 2^{-s_{m,i}}$.

2.2.1 Sistemas de Coordenadas

O resultado de qualquer transformação linear depende do sistema de coordenadas utilizado. No caso do CORDIC, o sistema de coordenadas é determinado pela variável m, e pode ser circular, linear ou hiperbólico. Um exemplo típico é o cálculo direto do seno e cosseno, que só podem ser obtidos com o sistema circular de coordenadas.

Para um melhor entendimento, vamos considerar um sistema de coordenadas polares. Sendo (x, y) coordenadas ortogonais de um ponto P, temos:

$$\begin{cases} R = \sqrt{x^2 + m \cdot y^2} \\ \alpha = \left(\frac{1}{\sqrt{m}}\right) \tan^{-1}\left(\frac{y\sqrt{m}}{x}\right) \end{cases} \qquad \begin{cases} x = R \cdot \cos\left(\alpha\sqrt{m}\right) \\ y = \left(\frac{R}{\sqrt{m}}\right) \cdot sen\left(\alpha\sqrt{m}\right) \end{cases} \tag{2.8}$$

A variável m irá definir a norma do vetor $R = (x, y)^T$, dada por $\sqrt{x^2 + m \cdot y^2}$ e o ângulo α, definido por $(\frac{1}{\sqrt{m}}) \cdot \tan^{-1}(\frac{y\sqrt{m}}{x})$. Quando $m = 1$, a trajetória da rotação é um círculo de função $x^2 + y^2 = 1$ e $\alpha = \tan^{-1}(\frac{y}{x})$. No sistema de coordenadas hiperbólico, a trajetória da rotação é uma função hiperbólica definida por $x^2 - y^2 = 1$ e $\alpha = \tanh^{-1}(\frac{y}{x})$, e no sistema de coordenadas linear, a trajetória é dada pela linha $x = 1$ e $\alpha = \frac{y}{x}$.

A tabela abaixo resume de forma sucinta cada sistema de coordenadas usado no CORDIC.

Tabela 2.2: Sistema de Coordenadas no CORDIC

Sistemas de Coordenadas			
Modo	$\boldsymbol{m}$	**Raio**	**Ângulo**
Circular	$m = 1$	$\sqrt{x^2 + y^2}$	$\tan^{-1}(\frac{y}{x})$
Linear	$m = 0$	x	$\frac{y}{x}$
Hiperbólico	m = -1	$\sqrt{x^2 - y^2}$	$\tanh^{-1}(\frac{y}{x})$

A idéia principal é transformar linearmente um ponto $P_i = (x_i, y_i)$ para um ponto $P_{i+1} = (x_{i+1}, y_{i+1})$ tal como:

$$\begin{cases} x_{i+1} = x_i + m \cdot \delta_i \cdot y_i \\ y_{i+1} = y_i - \delta_i \cdot x_i \end{cases} \tag{2.9}$$

O entendimento fica ainda mais explícito visualizando as figuras abaixo, contendo as trajetórias das rotações em cada sistema de coordenadas:

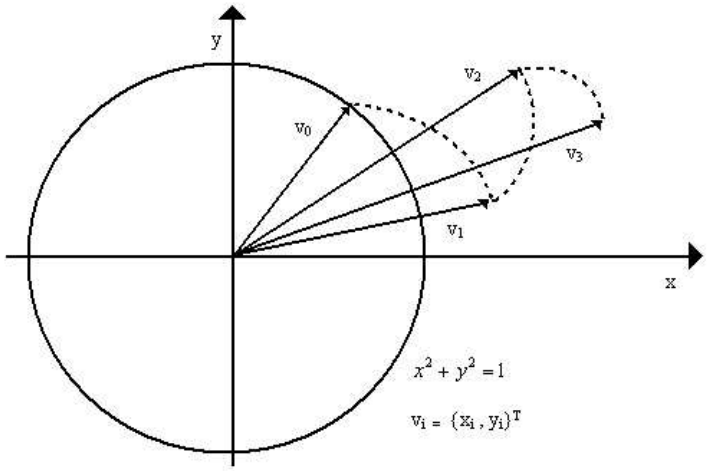

Figura 2.2: Sistema de Coordenadas Circular

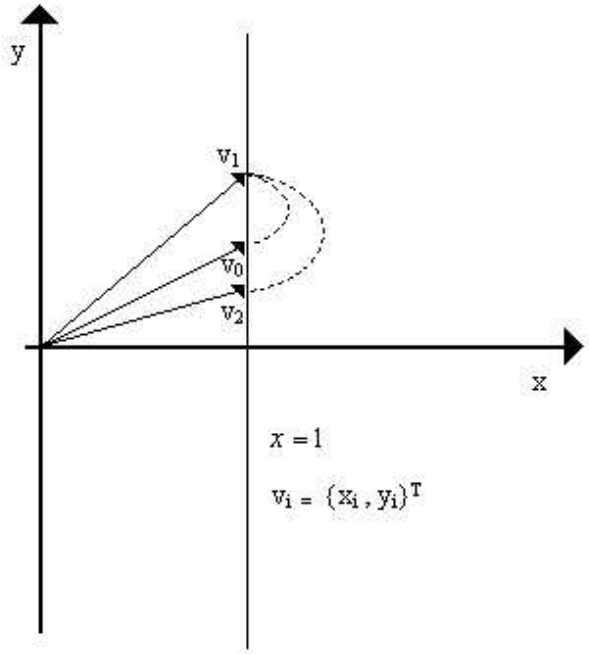

Figura 2.3: Sistema de Coordenadas Linear

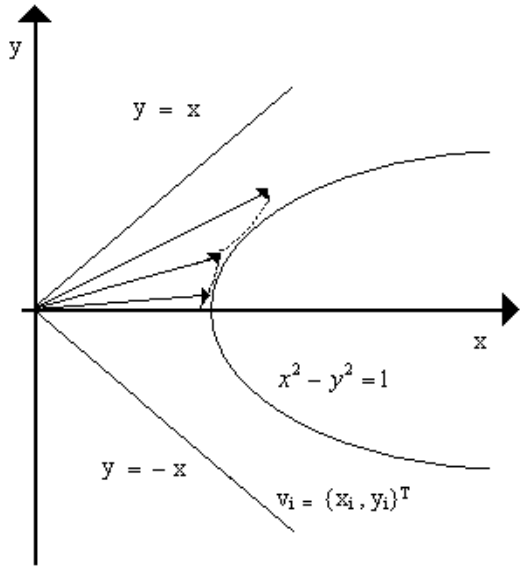

Figura 2.4: Sistema de Coordenadas Hiperbólico

Nas figuras citadas, o ângulo de rotação fixa $\alpha_{m,i}$ vai diminindo em módulo à medida que as iterações vão sendo incrementadas devido à seqüência de deslocamento. O efeito rotacional pode ocorrer tanto no sentido horário quanto no sentido anti-horário, de acordo com o esquema de controle de convergência do algoritmo. Sendo ϕ o somatório dos ângulos $\alpha_{m,i}$, temos:

$$\begin{cases} \alpha_{m,i} = \left(\frac{1}{\sqrt{m}}\right) \tan^{-1}\left(\sqrt{m}\delta_i\right) \\ \phi = \sum_{i=0}^{n-1} \mu_i \alpha_{m,i} \end{cases} \tag{2.10}$$

Pode-se observar nas Figuras 2.2 e 2.4 que o vetor vai sofrendo uma expansão não uniforme ao longo das iterações. Isto ocorre porque o algoritmo realiza rotações imperfeitas (multiplicações de senos e cossenos). Se acumularmos estas k_i expansões numa variável K_n, podemos corrigir o resultado através de um produto por um escalar, retirando o efeito das expansões. Este é o chamado Fator de Correção de Escala, sendo também um parâmetro para verificar se o algoritmo convergiu no sistema de coordenadas adotado.

$$\begin{cases} k_i = \sqrt{1 + m\delta_i^2} \\ K_n = \prod_{i=0}^{n-1} k_i \end{cases} \tag{2.11}$$

Uma terceira componente z_i armazena o ângulo total acumulado nas sucessivas iterações. Depois de n iterações, temos:

$$\begin{cases} z_{i+1} = z_i - \mu_i \cdot \alpha_{m,i} \\ z_n = z_0 - \sum_{i=0}^{n-1} \mu_i \cdot \alpha_{m,i} \end{cases} \tag{2.12}$$

Onde z_n é igual a diferença entre o ângulo inicial z_0 e o ângulo total de rotação acumulado nas n iterações.

2.3 Modos de Operação

Assim como o sistema de coordenadas, a escolha do modo de operação permite que se chegue a resultados específicos. Esta escolha é feita substituindo μ_i por um "esquema de controle", que determina a direção de cada rotação.

A variável μ_i pode assumir os valores {1,-1}, e quando o produto de μ_i e de x_i for positivo na equação (2.7), o sentido da rotação é anti-horário. Por sua vez, quando o produto destas variáveis for negativo, o sentido da rotação é horário.

Os modos de operação principais são os modos de rotação e o de vetorização.

2.3.1 Modo de Rotação

O modo de rotação, também conhecido como *Z-Reduction* e como *modo de aplicação*, caracteriza-se por ir reduzindo o valor da variável z a cada iteração convergindo a variável para zero. Quando isto acontece, o ângulo total de rotação acumulado é exatamente igual ao ângulo inicial z_0 escolhido.

Na prática, para aplicação do CORDIC num vetor de entrada $(x, y)^T$ é atribuído um ângulo inicial de rotação $z_0 = \theta$. Com isto, temos $x_0 = x$, $y_0 = y$ e $z_0 = \theta$, onde:

$$z_n = \theta - \sum_{i=0}^{n-1} \mu_i \cdot \alpha_{m,i} \tag{2.13}$$

Quando z_n é igual a zero, $\theta = \phi = \sum_{i=0}^{n-1} \mu_i \cdot \alpha_{m,i}$, ou seja, o ângulo total acumulado de rotação ϕ é igual a θ.

Para que isto aconteça, o esquema de controle $\mu_i = sign(z_i)$ é usado, obtendo-se:

$$\begin{cases} x_{i+1} = x_i - m \cdot sign(z_i) \cdot y_i \cdot 2^{-s_{m,i}} \\ y_{i+1} = y_i + sign(z_i) \cdot x_i \cdot 2^{-s_{m,i}} \\ z_{i+1} = z_i - sign(z_i) \cdot \alpha_{m,i} \end{cases} \tag{2.14}$$

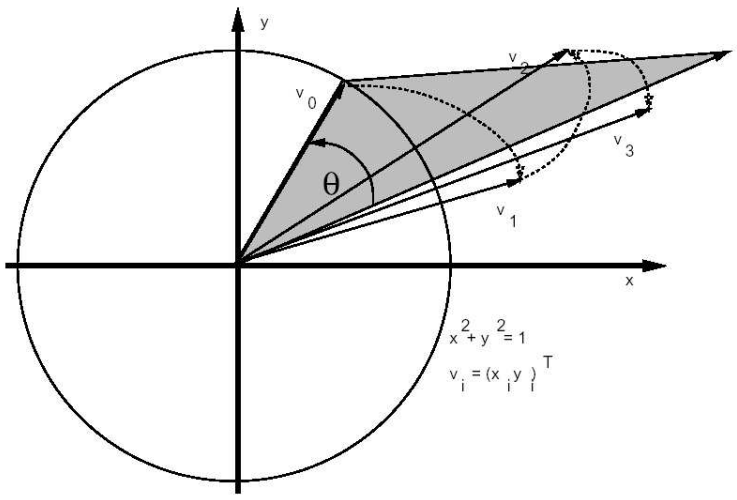

Figura 2.5: Trajetória de uma transformação CORDIC no modo de rotação com sistema de coordenadas circular.

Na Figura 2.5 é mostrada a trajetória para o modo de rotação no sistema de coordenadas circular. É evidente a expansão não uniforme a cada iteração, que pode ser corrigida com o fator de correção de escala.

2.3.2 Modo de Vetorização

Também conhecido como *Y-Reduction* e *modo de avaliação*, o *modo de vetorização* tem como finalidade rotacionar um vetor de entrada $(x, y)^T$ ao redor do eixo x. Isto é feito através de um esquema de controle capaz de forçar y_n para zero durante as iterações. A rotação ao redor do eixo x pode ser tanto no eixo positivo ($x_0 \geq 0$) quanto no eixo negativo ($x_0 \leq 0$), dependendo do sinal de x_0, e quando $y_n = 0$, z_n contém o ângulo total de rotação ϕ depois de n iterações.

$$z_n = -\phi = -\sum_{i=0}^{n-1} \mu_i \cdot \alpha_{m,i} \tag{2.15}$$

O esquema de controle no modo *Y-reduction* é $\mu_i = -sign(x_i) \cdot sign(y_i)$. Substituindo a variável μ_j na equação fundamental (2.7), temos:

$$\begin{cases} x_{i+1} = x_i + m \cdot sign(x_i) \cdot sign(y_i) \cdot y_i \cdot 2^{-s_{m,i}} \\ y_{i+1} = y_i - sign(x_i) \cdot sign(y_i) \cdot x_i \cdot 2^{-s_{m,i}} \\ z_{i+1} = z_i + sign(x_i) \cdot sign(y_i) \cdot \alpha_{m,i} \end{cases} \tag{2.16}$$

Ao final das iterações, $x_n = K_m(n) \cdot sign(x_0) \cdot \sqrt{x^2 + m \cdot y^2}$. Na figura abaixo pode-se observar a trajetória para o modo *Y-Reduction* no sistema de coordenadas circular de um vetor $(x, y)^T$.

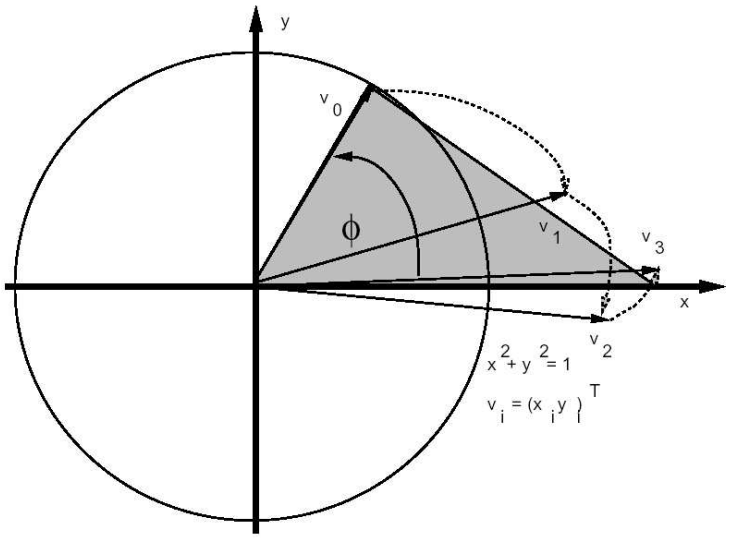

Figura 2.6: Trajetória de uma transformação CORDIC no modo Y-Reduction com sistema de coordenadas circular.

2.4 Convergência do Algoritmo

Na Seção 2.3 os esquemas de controle de convergência foram usados nos modos de rotação e vetorização. Os esquemas de controle definem uma seqüência de sinais $\mu_i \in \{1, -1\}$ para conduzir a direção das i iterações, mas não são os únicos responsáveis por garantir a convergência do CORDIC.

Sendo o número de iterações igual a n rotações de ângulos fixos com direção variável, o ângulo desejado de rotação A_0 pode apresentar um erro de arredondamento $\Delta\phi$ dado por:

$$\Delta\phi = A_0 - \sum_{i=0}^{n-1} \mu_i \cdot \alpha_{m,i} \tag{2.17}$$

O erro de arredondamento $\Delta\phi$ não inclui erro de quantização finita dos ângulos de rotação $\alpha_{m,i}$. Os erros de quantização são abordados por HU [8].

Analisando a equação (2.17), podemos definir dois critérios de convergência, conforme [2, 3]:

- O ângulo escolhido de rotação deve satisfazer à condição:

$$\alpha_{m,i} - \sum_{j=i+1}^{n-1} \alpha_{m,j} \leq \alpha_{m,n-1} \tag{2.18}$$

Exemplificando o caso limite descrito pela equação (2.18), se após uma determinada iteração i restar um ângulo A_i igual a zero (ou seja, o ângulo desejado para rotação foi atingido, mas ainda restam n-i iterações), significa que na próxima iteração o vetor bidimensional sofrerá uma rotação $\pm\alpha_{m,i}$. Neste caso, o somatório dos ângulos restantes de rotação $\sum_{j=i+1}^{n-1} \alpha_{m,j}$ seria suficiente para fazer com que o ângulo de rotação final A_n, após a última iteração n, seja zero com acurácia de $\alpha_{m,n-1}$.

- O ângulo de rotação A_0 não deve exceder à região de convergência da determinada iteração, ou seja, a soma de todos os ângulos restantes mais o ângulo final $\alpha_{m,n-1}$:

$$|A_0| \leq \sum_{i=0}^{n-1} \alpha_{m,i} + \alpha_{m,n-1} \tag{2.19}$$

2.4.1 Seqüências de Deslocamento

Os critérios expostos acima estão diretamente ligados à escolha de uma seqüência de deslocamento convergente, pois é ela que determina o ângulo fixo de cada rotação, dado por:

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} \cdot \tan^{-1}(\sqrt{m} \cdot 2^{-s_{m,i}}) \qquad \textit{sendo } i \in \{0, \ldots, n-1\} \tag{2.20}$$

Uma mesma seqüência de deslocamento nem sempre converge em diferentes sistemas de coordenadas, o que possibilita a atuação do algoritmo em diferentes

regiões de convergência, com fatores de correção de escala diferentes (equações (2.10) e (2.11)).

Assim que for escolhida a seqüência, independente do modo de operação, pode-se calcular antecipadamente os ângulos fixos $\alpha_{m,i}$ e o fator de correção de escala.

Para os sistemas de coordenadas circular e linear, é intuitiva a escolha do conjunto dos números naturais $\mathbb{N}$ para $s_{m,i}$, pois a seqüência seria inteira e não decrescente, fazendo com que $\delta_{m,i} = 2^{-s_{m,i}}$ sempre convirja. Já quando o sistema de coordenadas for hiperbólico, a seqüência anterior só convergirá se repetirmos algumas iterações para alcançar o resultado desejado. Esta idéia foi desenvolvida por Walther [3] e consiste em montar a seqüência $s_{m,i}$ com o conjunto $\mathbb{N}^*$ repetindo as iterações $3 \cdot k + 1$, sendo k igual às iterações $\{1, 2, \cdots, n\}$.

A tabela a seguir mostra as seqüências de deslocamento para cada sistema de coordenadas.

Tabela 2.3: Seqüência de Deslocamento do CORDIC

Sistemas de de Coordenadas	**Seqüência de de Deslocamento**	**Região de de Convergência**	**Fator de Correção de Escala**
m	$s_{m,i}$	$\lvert A_0 \rvert$	$K_m(n \to \infty)$
1	$0, 1, 2, 3, 4, \cdots, i, \cdots$	$\sim 1,74$	$\sim 1,64676$
0	$1, 2, 3, 4, \cdots, i, \cdots$	1,0	1,0
-1	$1, 2, 3, 4, 4, 5, \cdots$	$\sim 1,13$	$\sim 0,82816$

Visando aumentar a região de convergência para os sistemas de coordenadas linear, foi utilizada uma seqüência diferente da exposta por Meyr e Dawid [9] ao longo deste trabalho. A seqüência sugerida por eles não leva em consideração o número 0, que limita o ângulo máximo $|A_0|$ em 1 radiano (ver Tabela 2.3). Ao acrescentar o número 0 na seqüência, a região de convergência é simplesmente dobrada, conforme mostrado na tabela abaixo e na seção de implementação deste capítulo.

Tabela 2.4: Seqüência de Deslocamento modificada do CORDIC

Sistemas de de Coordenadas	**Seqüência de de Deslocamento**	**Região de de Convergência**	**Fator de Correção de Escala**
m	$s_{m,i}$	$\|A_0\|$	$K_m(n \to \infty)$
0	$0, 1, 2, 3, 4, \cdots, i, \cdots$	2,0	1,0

2.5 Funções CORDIC

Para qualquer implementação CORDIC, é necessário conhecer o sistema de coordenadas (circular, linear ou hiperbólico), o modo de operação e a seqüência de deslocamento. Com estes três graus de liberdade, diferentes funções podem ser calculadas a partir de um vetor de entrada escolhido (x_0, y_0, z_0).

Para um melhor entendimento, a tabela 2.5 mostra a permutação das combinações acima e as respectivas funções obtidas em cada modo de operação.

Tabela 2.5: Funções calculadas pelo CORDIC

m	**Modo de Operação**	**Entrada**	**Saída**	**Funções**
1	rotação	$x_0 = x$	$x_n = K_1(n) \cdot (x\cos\theta - ysen\theta)$	tan
		$y_0 = y$	$y_n = K_1(n) \cdot (y\cos\theta + xsen\theta)$	sen
		$z_0 = \theta$	$z_n = 0$	cos
		$x_0 = \frac{1}{K_1(n)}$	$x_n = \cos\theta$	csc
		$y_0 = 0$	$y_n = sen\theta$	sec
		$z_0 = \theta$	$z_n = 0$	
1	vetorização	$x_0 = x$	$x_n = K_1(n) \cdot sign(x_0) \cdot \sqrt{x^2 + y^2}$	atan
		$y_0 = y$	$y_n = 0$	
		$z_0 = \theta$	$z_n = \theta + \tan^{-1}(\frac{y}{x})$	
0	rotação	$x_0 = x$	$x_n = x$	multiplicação
		$y_0 = y$	$y_n = y + x \cdot z$	soma
		$z_0 = z$	$z_n = 0$	
0	vetorização	$x_0 = x$	$x_n = x$	divisão
		$y_0 = y$	$y_n = 0$	
		$z_0 = z$	$z_n = z + \frac{y}{x}$	
-1	rotação	$x_0 = x$	$x_n = K_{-1}(n) \cdot (x\cosh\theta + ysenh\theta)$	tanh
		$y_0 = y$	$x_n = K_{-1}(n) \cdot (y\cosh\theta + xsenh\theta)$	senh
		$z_0 = \theta$	$z_n = 0$	cosh
		$x_0 = \frac{1}{K_{-1}(n)}$	$x_n = \cosh\theta$	exp
		$y_0 = 0$	$y_n = senh\theta$	
		$z_0 = \theta$	$z_n = 0$	
-1	vetorização	$x_0 = x$	$x_n = K_{-1}(n) \cdot sign(x_0) \cdot \sqrt{x^2 - y^2}$	atanh
		$y_0 = y$	$y_n = 0$	sqrt
		$z_0 = \theta$	$z_n = \theta + \tanh^{-1}(\frac{y}{x})$	ln

Um exemplo da utilização da tabela acima é a implementação da função seno CORDIC. A função seno pode ser obtida no sistema de coordenadas circular, modo Z-Reduction com o vetor de entrada sendo $(x_0 = \frac{1}{K_1(n)}, y_0 = 0, z_0 = \theta)$. Supondo $\theta = 68°$, com 5 bits de precisão, temos a seqüência de rotações a seguir:

Tabela 2.6: Exemplo das iterações CORDIC para o cálculo do seno de 68°

Iteração	**Entrada**	**Saída**	**Parâmetros CORDIC**
1	$x_0 = 1$	$x_1 = 0.7071$	$\alpha_1 = 45°$
	$y_0 = 0$	$y_1 = 0.7071$	$K_1 = 1,4142$
	$z_0 = 68°$	$z_1 = 23°$	$\mu_1 = +1$
2	$x_1 = 0.7071$	$x_2 = 0.3162$	$\alpha_2 = 26.5°$
	$y_1 = 0.7071$	$y_2 = 0.9487$	$K_2 = 1.1180$
	$z_1 = 23°$	$z_2 = -3.5°$	$\mu_2 = +1$
3	$x_2 = 0.3162$	$x_3 = 0.5369$	$\alpha_3 = 14°$
	$y_2 = 0.9487$	$y_3 = 0.8437$	$K_3 = 1.0308$
	$z_2 = -3.5°$	$z_3 = 10.5°$	$\mu_3 = -1$
4	$x_3 = 0.5369$	$x_4 = 0.4281$	$\alpha_4 = 7°$
	$y_3 = 0.8437$	$y_4 = 0.9037$	$K_4 = 1.0078$
	$z_3 = 10.5°$	$z_4 = 3.3°$	$\mu_4 = +1$
5	$x_4 = 0.4281$	$x_5 = 0.3709$	$\alpha_5 = 3.5°$
	$y_4 = 0.9037$	$y_5 = 0.9287$	$K_5 = 1.0020$
	$z_4 = 3.3°$	$z_5 = -0.2°$	$\mu_5 = +1$

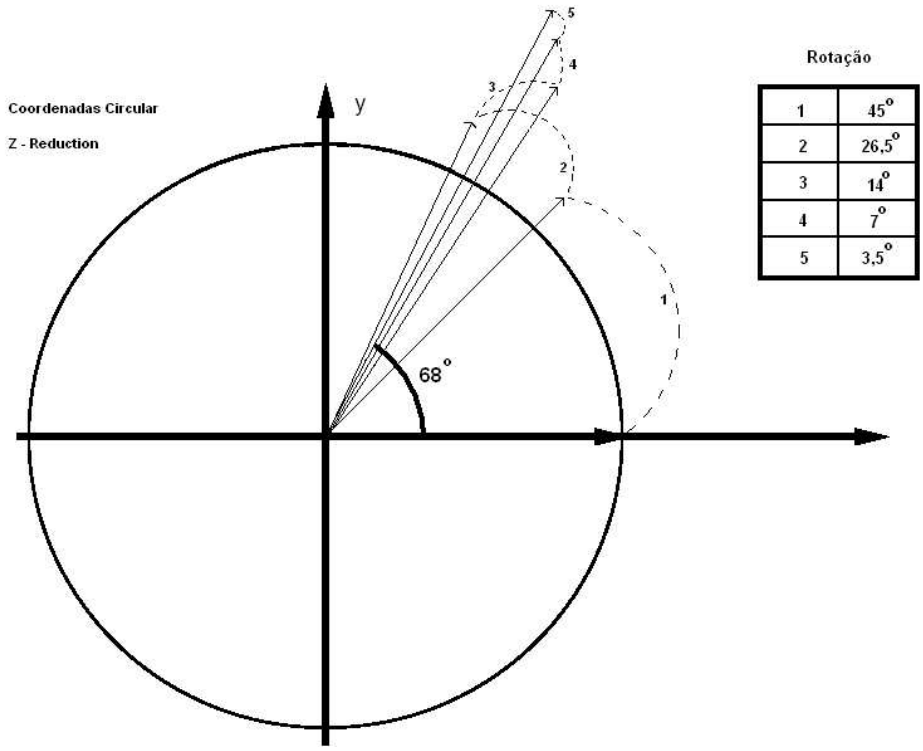

Figura 2.7: Trajetória das rotações CORDIC no cálculo do seno de 68°.

A Figura 2.7 mostra a trajetória das rotações CORDIC no cálculo do seno de 68°. A variável z, iniciada com $\theta = 68°$, converge para o valor 0, ou seja, atinge o ângulo desejado. O valor do seno pode ser visto na variável y, que está ilustrada sem a aplicação do Fator de Correção de Escala, dando ênfase nas imperfeições do algoritmo CORDIC.

2.6 Fator de Correção de Escala

Como podemos observar nas Tabelas 2.3 e 2.5, quando o sistema de coordenadas é circular ou hiperbólico é necessário corrigir a saída das variáveis x e y do vetor para que seja possível obter o resultado esperado. Isto ocorre porque a cada iteração CORDIC essas variáveis sofrem uma expansão não uniforme. Vejamos o exemplo quando $m = 1$, em que:

$$\begin{cases} x_n = K_1(n) \cdot (x\cos\theta - ysen\theta) \\ y_n = K_1(n) \cdot (y\cos\theta + xsen\theta) \end{cases} \tag{2.21}$$

O valor de x_n e y_n está multiplicado por $K_1(n)$ (vide equação (2.11)). Para se obter o valor correto das variáveis desejadas é preciso multiplicá-las por um *Fator de Correção de Escala*, sendo $K_{FCE} = \frac{1}{K_1(n)}$. Neste caso:

$$\begin{cases} x_n = K_{FCE} \cdot K_1(n) \cdot (x\cos\theta - ysen\theta) \\ y_n = K_{FCE} \cdot K_1(n) \cdot (y\cos\theta + xsen\theta) \end{cases} \tag{2.22}$$

O fator de correção depende do sistema de coordenadas e da seqüência de deslocamento, podendo ser pré-calculado antes das iterações CORDIC para ser usado diretamente como entrada, evitando multiplicações escalares *a posteriori*.

Numa primeira análise, o fator de correção de escala pode não ser motivo para maiores preocupações, mas pelo fato de requerer pelo menos duas multiplicações ou duas divisões no produto final, é alvo de estudo por vários pesquisadores que visam tornar o CORDIC cada vez mais rápido. Como exemplo, há os métodos de Multiplicação Constante do Fator [10], de Passos de Normalização [11], de Iterações

Repetidas e de Iterações Compensadas [12], dentre outros. Neste trabalho, todas as simulações foram baseadas na Multiplicação Constante do Fator, que é a característica de usar o fator na entrada do vetor $(x, y)^T$.

2.7 Acurácia do Algoritmo

Através de n rotações de ângulos fixos um dado ângulo A_0 é calculado, com a incerteza de ser aproximado por um erro $\Delta\phi \leq \alpha_{m,n-1}$. Mesmo se todas as outras fontes de erro forem desprezadas, a acurácia é limitada em magnitude pelo último ângulo de rotação $\alpha_{m,n-1}$. À medida que n aumenta, aproximadamente $s_{m,n-1}$ dígitos de acurácia são obtidos desde que $s_{m,n-1}$ represente o último número da seqüência de deslocamento. Portanto, para uma acurácia de W bits, o número de iterações deve ser escolhido tal que $s_{m,n-1} = W$.

Uma segunda fonte de erro ocorre devido à precisão finita das variáveis envolvidas, seja numa implementação de ponto fixo ou numa implementação de ponto flutuante. Dados W bits de entrada ao tamanho da palavra (ponto fixo), aumentado por G bits de guarda do bit mais significativo (MSB) e C bits de guarda para o bit menos significativo (LSB) [9]. Os bits de guarda MSB são necessários no sistema circular ($m = 1$), pois o fator de correção de escala é maior que 1. Já os bits de guarda LSB são necessários devido aos eventuais erros de aproximação $\Delta\phi$.

A acurácia do CORDIC foi a principal motivação de sua empregabilidade neste trabalho, pois permite que alguns resultados sejam alcançados com operações e cálculos matemáticos simplificados. Aplicações de álgebra linear, assim como na área de comunicações móveis onde o processamento nas estações rádio-base demandam cada vez mais velocidade e eficiência, são exemplos potencias para o uso do algoritmo. Através de suas transformações, cálculos computacionais que envolvam matrizes/vetores de ordem considerável podem ser simplificados. Como uma calculadora "clássica", operações matemáticas podem ser aproximadas com CORDIC, com precisão escolhida pelo próprio usuário. É por isto que o CORDIC foi explorado ao longo desta tese.

2.8 Implementação: CORDIC no Matlab

Todas as funções CORDIC demonstradas na Tabela 2.5 foram implementadas no Matlab, respeitando o sistema de coordenadas e o modo de operação correspondente.

Para comprovar o funcionamento da implementação, o fator de correção de escala e o ângulo total rotacionado (parâmetros do CORDIC) podem ser saídas de cada função, o que permite ao final da execução uma rápida verificação da convergência do algoritmo. O comando help [função] pode ser usado à vontade, facilitando a simulação.

Conforme explicado na Seção 2.4, cada função tem a sua região de convergência, que determina a faixa de operação do algoritmo. Esta desvantagem aparente do CORDIC não tira a sua importância, pois os vetores de entrada podem ser normalizados.

Os esquemas de controle de convergência μ_i para cada modo de operação (ver equação (2.7)) foram aplicados com o auxílio de uma função criada a partir da função *sign*, do Matlab. Esta não atende aos requisitos necessários por fazer $sign(0) = 0$. A solução foi adaptar a função já existente chamando-a de *bsign*, que faz $sign(0) = 1$. A função *bsign* foi empregada em todas as funções CORDIC implementadas.

Outro ponto importante para fixação do conceito foi estudar o comportamento de cada variável das funções CORDIC criadas, através de gráficos comparando o resultado obtido com n bits de precisão com o resultado obtido pelo Matlab. A região de convergência de cada função, crucial para correta aplicação do CORDIC, também pode ser melhor observada através destes gráficos. Cada gráfico possui o modelo de distribuição abaixo:

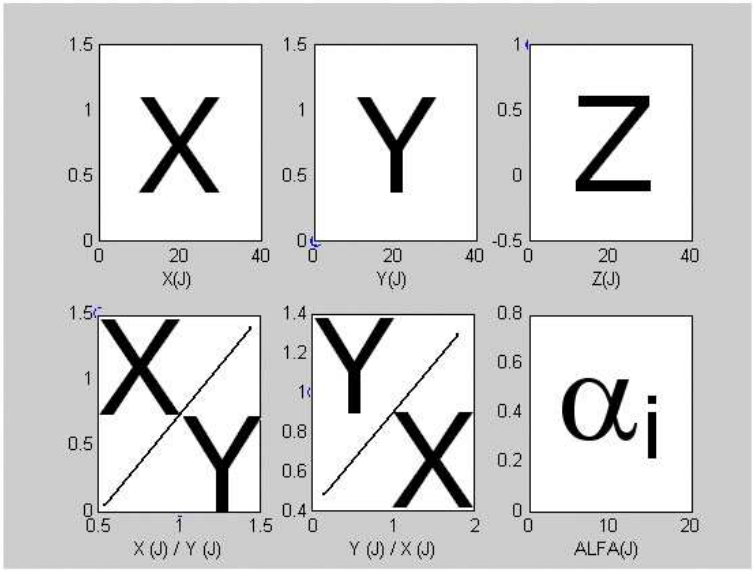

Figura 2.8: Modelo dos gráficos

Os gráficos ilustram o comportamento das variáveis x, y, z, $\frac{x}{y}$, $\frac{y}{x}$, e α_i ao longo das iterações, sendo α_i o ângulo rotacionado na i-ésima iteração CORDIC.

É importante frisar que as variáveis x e y estão refletidas nos gráficos após sofrerem a correção do fator de escala ($\frac{1}{K_n}$), no caso dos sistemas de coordenadas circulares e hiperbólicos. Como sugestão, é importante acompanhar a análise dos gráficos em conjunto com a Tabela 2.5. Há de se ressaltar que os gráficos das variáveis $\frac{x}{y}$ e $\frac{y}{x}$ só terão utilidade para as funções tangente e tangente hiperbólica, sendo excesso de informação para as demais funções. Somente no caso especial das funções secante e cossecante os gráficos das variáveis $\frac{x}{y}$ e $\frac{y}{x}$ serão substituídos por $\frac{1}{x}$ e $\frac{1}{y}$.

2.8.1 Função Tangente Cordic (tan_c)

A função tangente cordic é obtida com o sistema de coordenadas circular ($m = 1$), no modo de operação Z-Reduction.

No gráfico a seguir podemos ver a tangente cordic de 1 radiano com 20 bits de precisão. Seu resultado é $1,5574$, exatamente o mesmo resultado da função tangente do Matlab, como podemos observar na posição da variável $\frac{y}{x}$. Conforme esperado (ver Tabela 2.3), o seu sistema de coordenadas e sua seqüência de deslocamento determinam $K_n = 1,6468$ e $|A_0| = 1,74$. A acurácia desta implementação é de 20

bits.

Podemos acompanhar a redução da variável z em cada iteração, até se aproximar do valor limite de zero (Z-Reduction). Isto mostra a convergência do algoritmo, que no caso desta função pode ser aplicado para qualquer ângulo em radianos, uma vez que o maior ângulo de rotação possível devido à seqüência de deslocamento é $|A_0| = 1,74$, cobrindo todo o primeiro quadrante.

Notamos ainda que a variável α_i vai tendendo a zero com o andamento das rotações, sendo uma seqüência fixa para cada sistema de coordenadas. O gráfico desta variável repetir-se-á em todas as funções obtidas através do sistema de coordenadas circular.

Tangente CORDIC	
Comando	tan_c(1,20)
Coordenadas	Circular
Modo de Operação	Z-Reduction
Resultado CORDIC	1,5574
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.9: Tangente Cordic

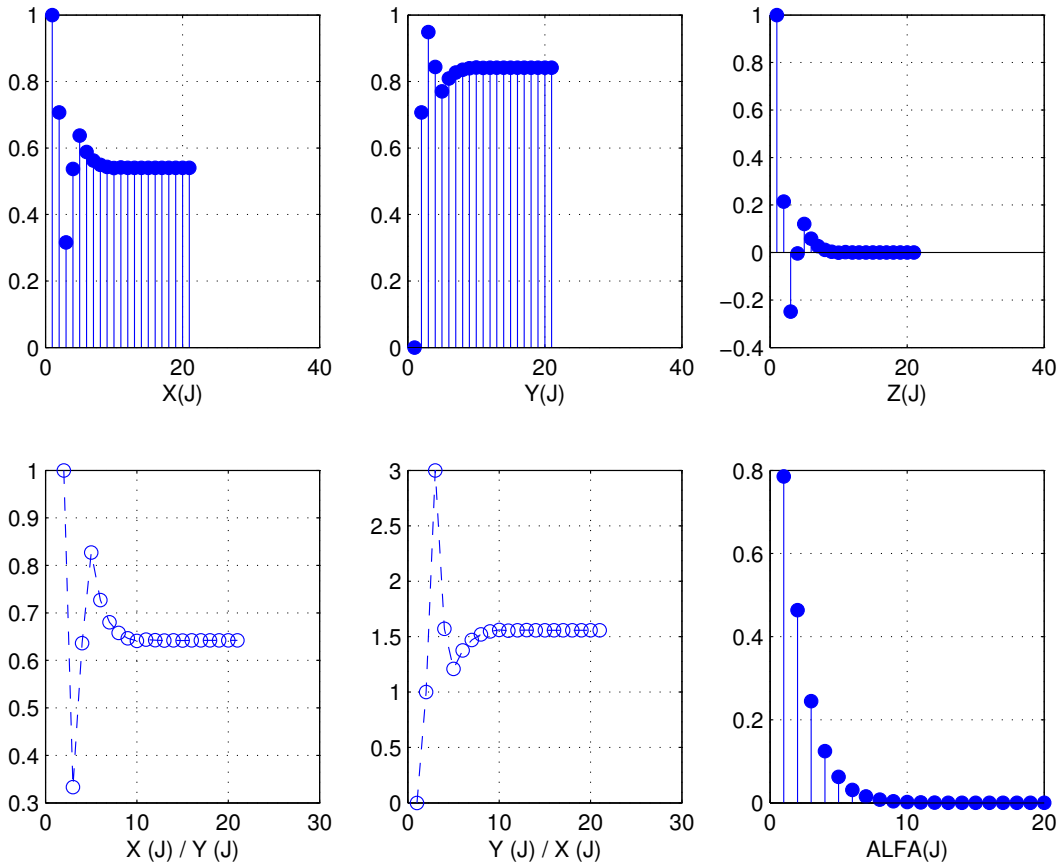

Figura 2.10: Gráfico da função Tangente Cordic

2.8.2 Função Tangente Inversa Cordic ($atan\_c$)

A função tangente inversa cordic também é obtida com o sistema de coordenadas circular ($m = 1$), mas no modo de operação Y-Reduction.

No gráfico da variável z abaixo podemos ver a tangente inversa cordic de 1 radiano com 20 bits de precisão.

A cada iteração é possível acompanhar a redução da variável y, até atingir o valor limite de zero (Y-Reduction).

Tangente Inversa CORDIC	
Comando	atan_c(1,20)
Coordenadas	Circular
Modo de Operação	Y-Reduction
Resultado CORDIC	0,7854
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.11: Tangente Inversa Cordic

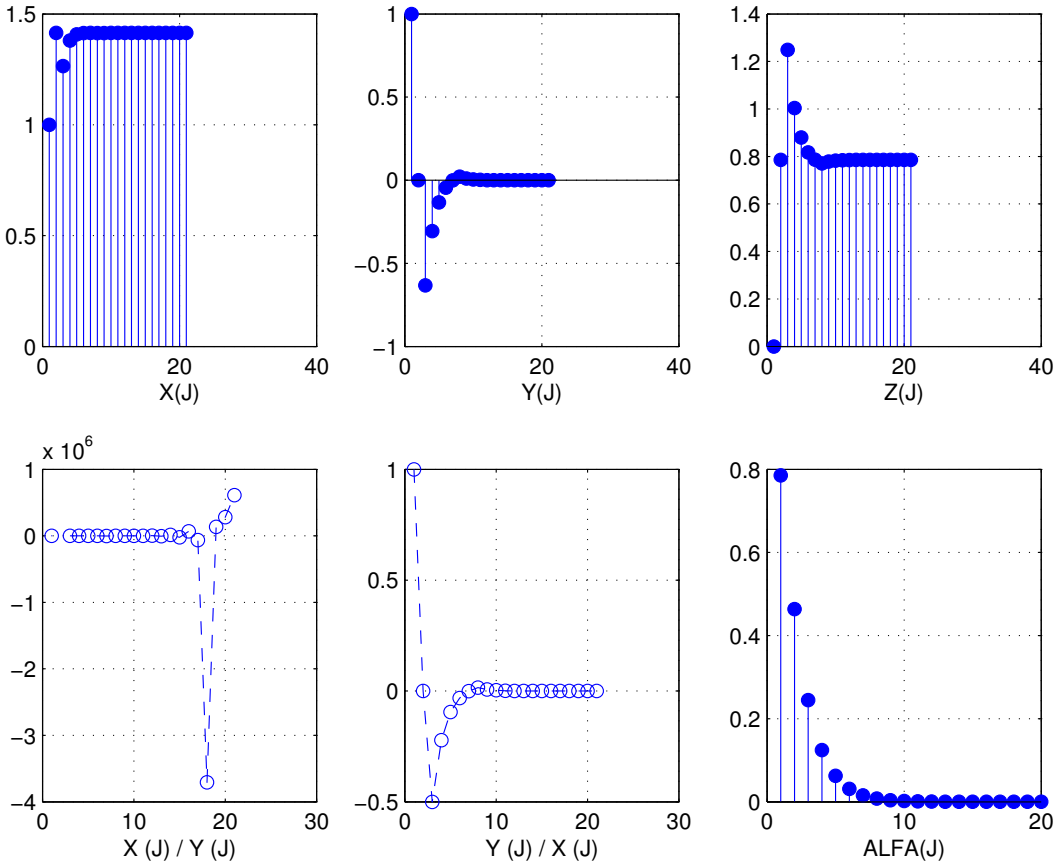

Figura 2.12: Gráfico da função Tangente Inversa Cordic

2.8.3 Função Cosseno Cordic (cos_c)

A função cosseno cordic é obtida com o sistema de coordenadas circular ($m = 1$), no modo de operação Z-Reduction.

No gráfico a seguir podemos ver o cosseno cordic de 1 radiano com 20 bits de precisão, como observamos na posição da variável x (o fator de correção de escala $\frac{1}{K_n}$ foi aplicado a cada iteração).

Cosseno CORDIC	
Comando	cos_c(1,20)
Coordenadas	Circular
Modo de Operação	Z-Reduction
Resultado CORDIC	0,5403
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.13: Cosseno Cordic

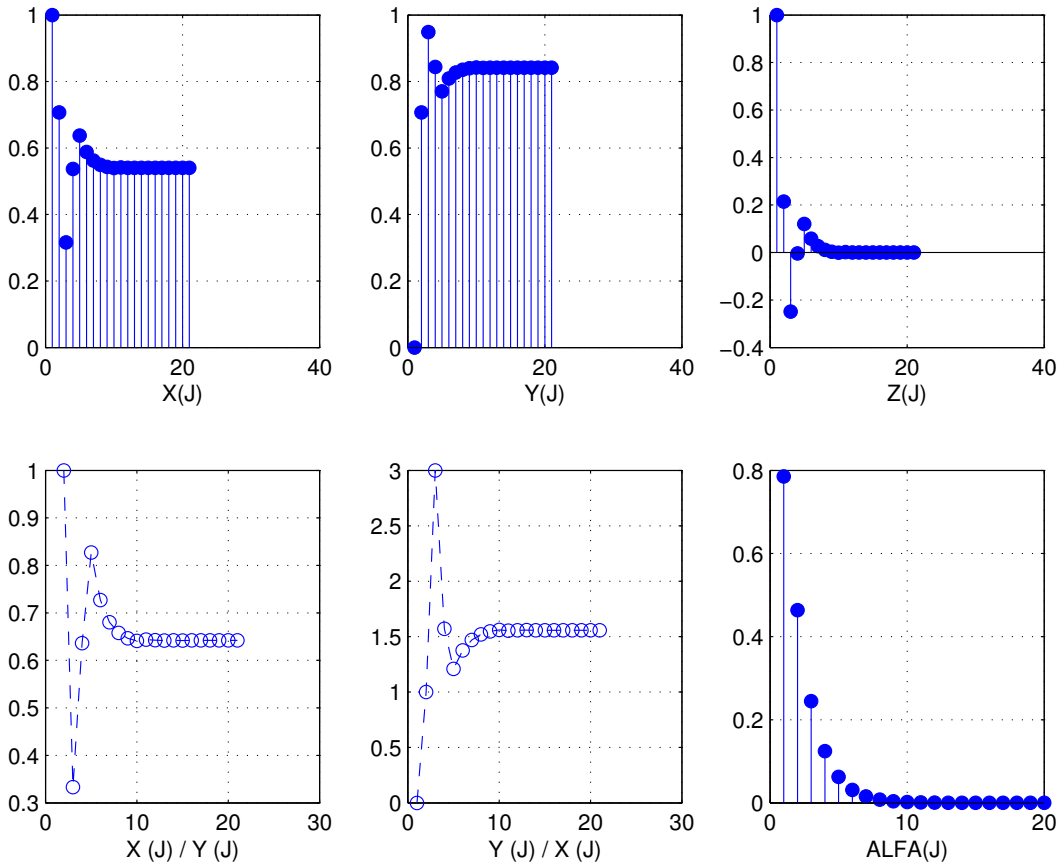

Figura 2.14: Gráfico da função Cosseno Cordic

2.8.4 Função Seno Cordic (sin_c)

A função seno cordic é obtida com o sistema de coordenadas circular ($m = 1$), no modo de operação Z-Reduction.

A Figura 2.14 gerada para o cosseno cordic é o mesmo gráfico gerado para a função seno cordic, no caso de 1 radiano com 20 bits de precisão, mas a variável de interesse agora é a variável y (o fator de correção de escala $\frac{1}{K_n}$ foi multiplicado a cada iteração).

Seno CORDIC	
Comando	sin_c(1,20)
Coordenadas	Circular
Modo de Operação	Z-Reduction
Resultado CORDIC	0,8415
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.15: Seno Cordic

2.8.5 Função Secante Cordic (sec_c)

A função secante cordic é obtida com o sistema de coordenadas circular ($m = 1$), no modo de operação Z-Reduction.

No gráfico abaixo podemos ver a secante cordic de 1 radiano com 20 bits de precisão, na posição da variável $\frac{1}{x}$ (já com a multiplicação do fator de correção de escala).

Secante CORDIC	
Comando	sec_c(1,20)
Coordenadas	Circular
Modo de Operação	Z-Reduction
Resultado CORDIC	1,8508
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.16: Secante Cordic

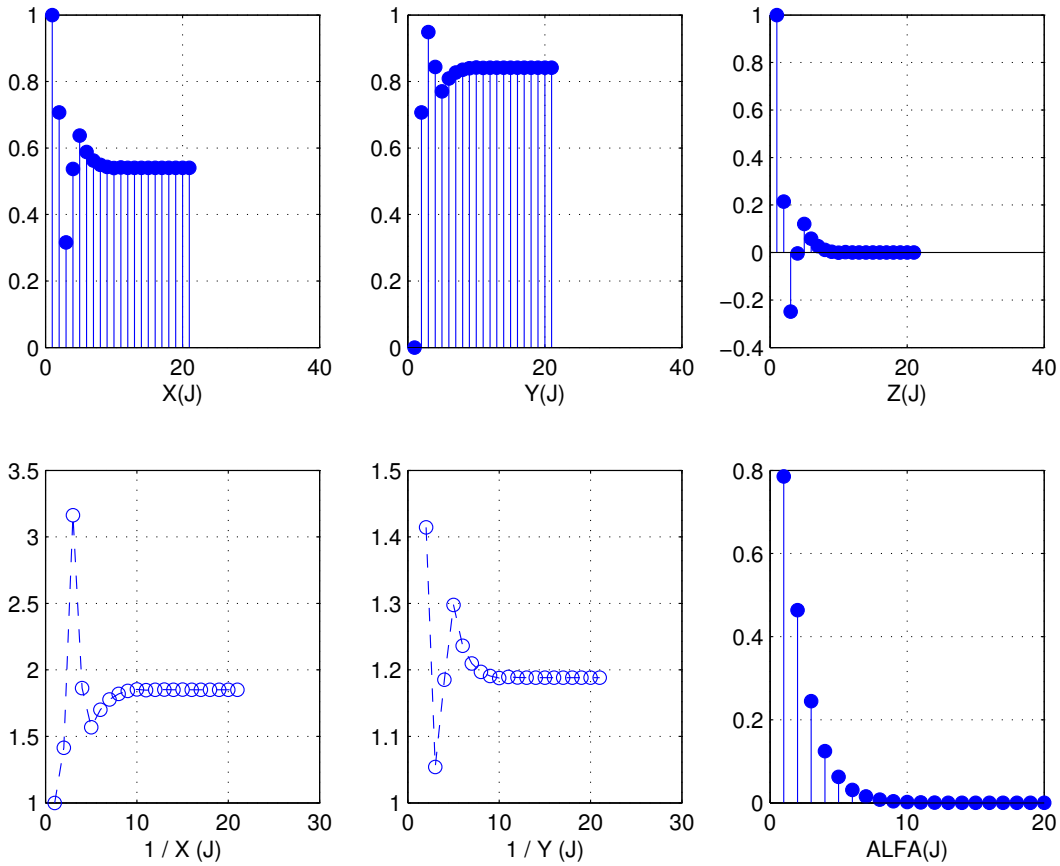

Figura 2.17: Gráfico da função Secante Cordic

2.8.6 Função Cossecante Cordic (csc_c)

A função cossecante cordic é obtida com o sistema de coordenadas circular ($m = 1$), no modo de operação Z-Reduction.

No gráfico acima (Figura 2.17, mesma gerada para a secante cordic), podemos ver a cossecante cordic de 1 radiano com 20 bits de precisão na posição da variável $\frac{1}{y}$, com o fator de correção de escala já aplicado.

Cossecante CORDIC	
Comando	csc_c(1,20)
Coordenadas	Circular
Modo de Operação	Z-Reduction
Resultado CORDIC	1,1884
Kn	1,6468
$\|A_0\|$	1,74
Acurácia	20 bits
Convergência	Qualquer ângulo

Figura 2.18: Cossecante Cordic

2.8.7 Função Soma Cordic (*soma_c*)

A função soma cordic é obtida com o sistema de coordenadas linear ($m = 0$), no modo de operação Z-Reduction.

No gráfico a seguir, podemos ver a soma cordic de 13 unidades com 127 unidades com 20 bits de precisão, como observamos na posição da variável y. Conforme esperado (ver Tabela 2.3), o seu sistema de coordenadas e sua seqüência de deslocamento determinam $K_n = 1$ e $|A_0| = 2$. A acurácia desta implementação é de 2^{-20}.

O algoritmo converge para soma de quaisquer números reais. Isto porque a variável z_0 na entrada da função foi pré-determinada a ser sempre 1 (ver Tabela 2.5).

A variável z_n converge a 0, devido ao modo de operação CORDIC ser o modo Z-Reduction.

Notamos ainda que a variável α_i assume uma seqüência de ângulos de rotações diferentes dos apresentados nas funções com sistema de coordenadas circular. Este gráfico repetir-se-á sempre que o sistema de coordenadas for linear.

Soma CORDIC	
Comando	soma_c(13,127,20)
Coordenadas	Linear
Modo de Operação	Z-Reduction
Resultado CORDIC	140,0000
Kn	1,0
$\|A_0\|$	2,0
Acurácia	20 bits
Convergência	Qualquer Nº Real

Figura 2.19: Soma Cordic

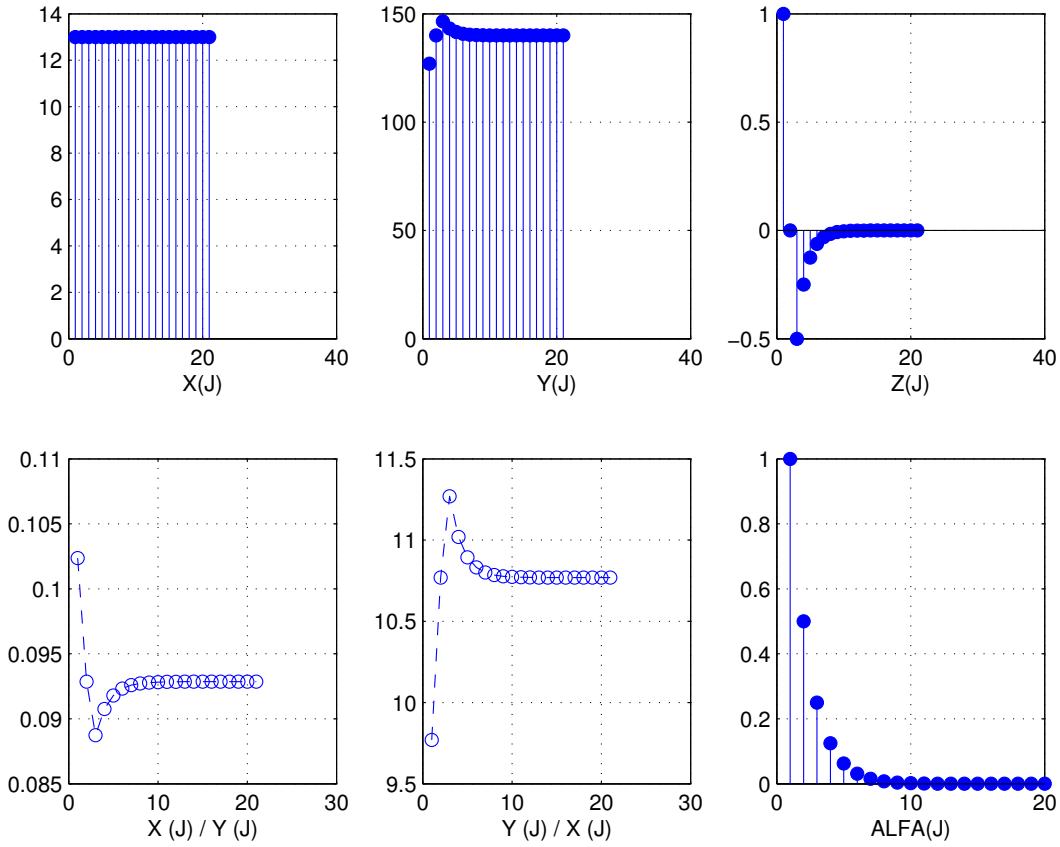

Figura 2.20: Gráfico da função Soma Cordic

2.8.8 Função Multiplicação Cordic ($mult\_c$)

A função multiplicação cordic é obtida com o sistema de coordenadas linear ($m = 0$), no modo de operação Z-Reduction.

No gráfico a seguir, podemos ver a multiplicação cordic de 13 unidades com 1,3 unidades com 20 bits de precisão, como observamos na posição da variável y. Esta variável é iniciada com o valor 0 ($y_0 = 0$) e armazena o valor final da multiplicação de x por z.

O algoritmo converge para multiplicação de quaisquer números reais desde que $z \leq 2$. Isto porque o maior ângulo de rotação do algoritmo (A_0) possui valor 2, que é o valor máximo que pode assumir a variável de entrada z_0, devido à seqüência de deslocamento.

Multiplicação CORDIC	
Comando	mult_c(13,1.3,20)
Coordenadas	Linear
Modo de Operação	Z-Reduction
Resultado CORDIC	16,9
Kn	1,0
$\|A_0\|$	2,0
Acurácia	20 bits
Convergência	$z \leq 2$

Figura 2.21: Multiplicação Cordic

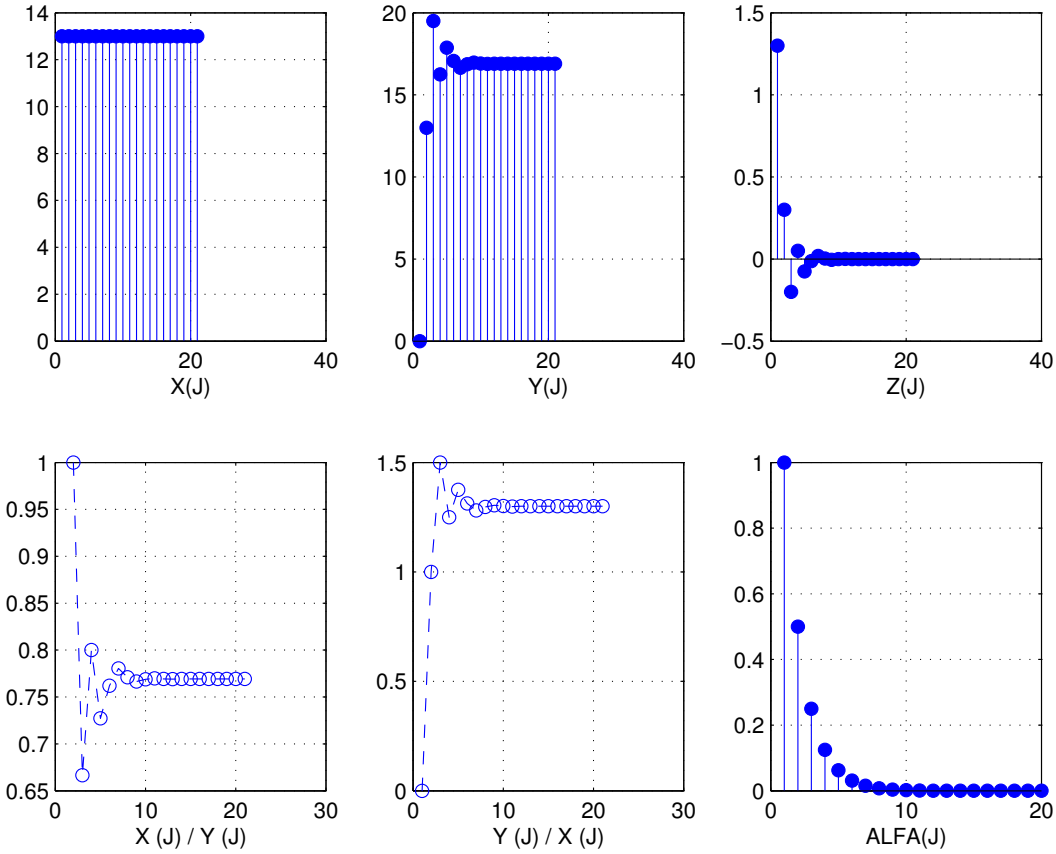

Figura 2.22: Gráfico da função Multiplicação Cordic

2.8.9 Função Divisão Cordic (div_c)

A função divisão cordic é obtida com o sistema de coordenadas linear ($m = 0$), no modo de operação Y-Reduction.

No próximo gráfico, podemos ver a divisão cordic de 13 unidades com 7 unidades com 20 bits de precisão, como observamos na posição da variável z. Esta variável é iniciada com o valor 0 ($z_0 = 0$) e armazena o valor final da divisão de y por x.

O algoritmo converge para divisão de quaisquer números reais desde que $y \leq x \cdot 2$. Isto porque o maior ângulo de rotação do algoritmo (A_0) possui valor 2, para satisfazer a segunda condição de convergência do CORDIC como visto na equação (2.19).

Divisão CORDIC	
Comando	div_c(13,7,20)
Coordenadas	Linear
Modo de Operação	Y-Reduction
Resultado CORDIC	1,8571
Kn	1,0
$\|A_0\|$	2,0
Acurácia	20 bits
Convergência	$y \leq 2.x$

Figura 2.23: Divisão Cordic

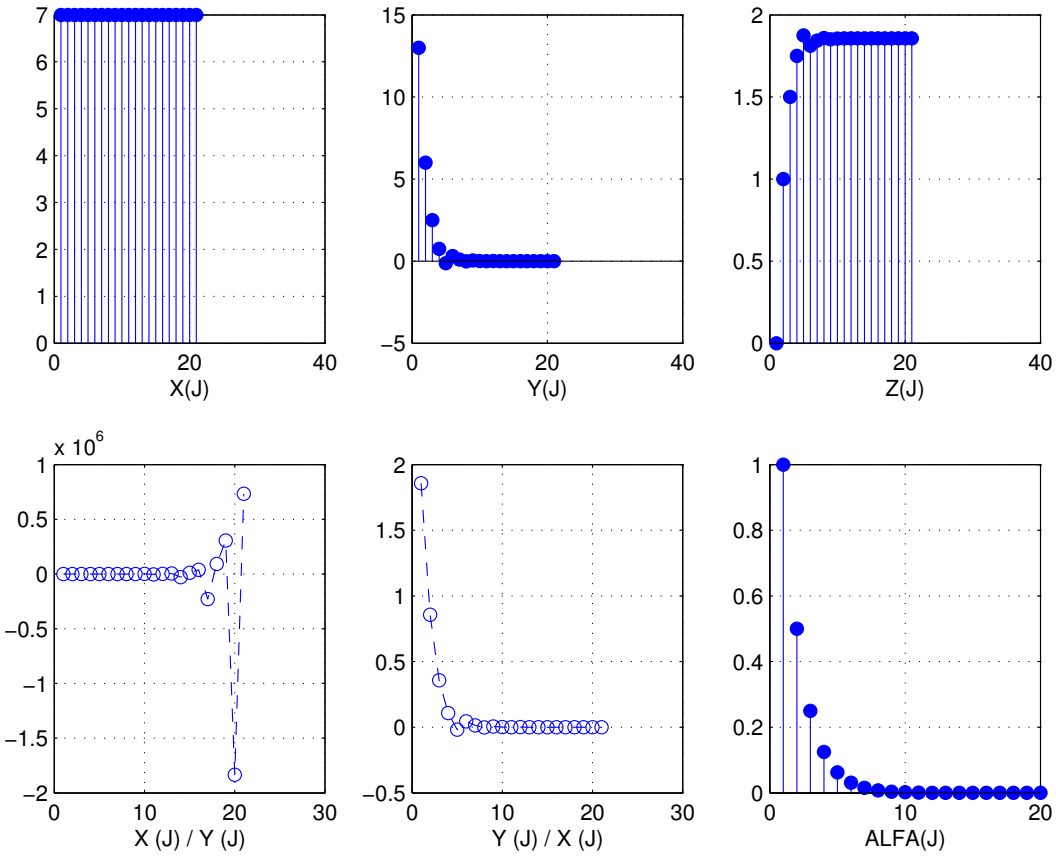

Figura 2.24: Gráfico da função Divisão Cordic

2.8.10 Função Tangente Hiperbólica Cordic ($tanh\_c$)

A função tangente hiperbólica cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Z-Reduction.

No gráfico a seguir, podemos ver a tangente cordic de 1 radiano com 20 bits de precisão, como observamos na posição da variável $\frac{y}{x}$. Conforme esperado (ver Tabela 2.3), o seu sistema de coordenadas e sua seqüência de deslocamento determinam $K_n = 0,8282$ e $|A_0| = 1,1182$. A acurácia desta implementação é de 2^{-20}.

Podemos acompanhar a redução da variável z a cada iteração, até atingir o valor limite de zero (Z-Reduction). Isto mostra a convergência do algoritmo, que no caso desta função, pode ser aplicado para qualquer ângulo em radianos menor do que 1,1181, uma vez que o maior ângulo de rotação possível dado pela seqüência de deslocamento é $|A_0| = 1,1182$.

Ainda percebemos que a variável α_i vai tendendo a zero com o andamento das rotações, sendo uma seqüência fixa para cada sistema de coordenadas. O gráfico desta variável repetir-se-á em todas as funções que forem coordenadas hiperbólicas.

Tangente Hiperbólica CORDIC	
Comando	tanh_c(1,20)
Coordenadas	Hiperbólico
Modo de Operação	Z-Reduction
Resultado CORDIC	0,7616
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Ângulos ≤ 1,1181

Figura 2.25: Tangente Hiperbólica Cordic

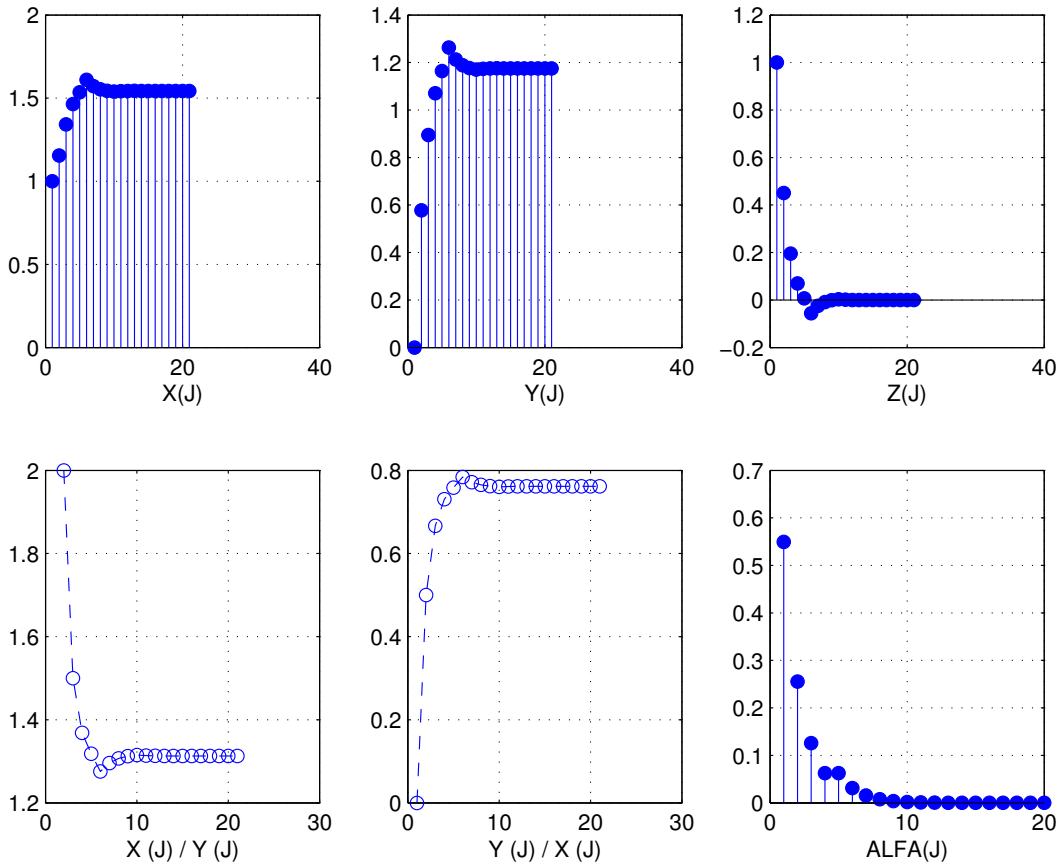

Figura 2.26: Gráfico da função Tangente Hiperbólica Cordic

2.8.11 Função Tangente Inversa Hiperbólica Cordic ($atanh\_c$)

A função tangente inversa hiperbólica cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Y-Reduction.

No gráfico a seguir, podemos ver a tangente inversa hiperbólica cordic de 0,7 radiano com 20 bits de precisão, conforme observado na posição da variável z. A entrada z_0 foi iniciada com o valor zero, para que o resultado seja obtido diretamente do valor final de z_n (ver Tabela 2.5).

Neste caso, o algoritmo converge para $\theta \leq 0,8069$, porque o arco cuja tangente hiperbólica é 0,8069 é arco 1,1182 radianos, que é o A_0 máximo de convergência dada pela seqüência de deslocamento.

Tangente Inversa Hiperbólica CORDIC	
Comando	atanh_c(0.7,20)
Coordenadas	Hiperbólico
Modo de Operação	Y-Reduction
Resultado CORDIC	0,8673
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Arcos $\leq$ 0,8069

Figura 2.27: Tangente Inversa Hiperbólica Cordic

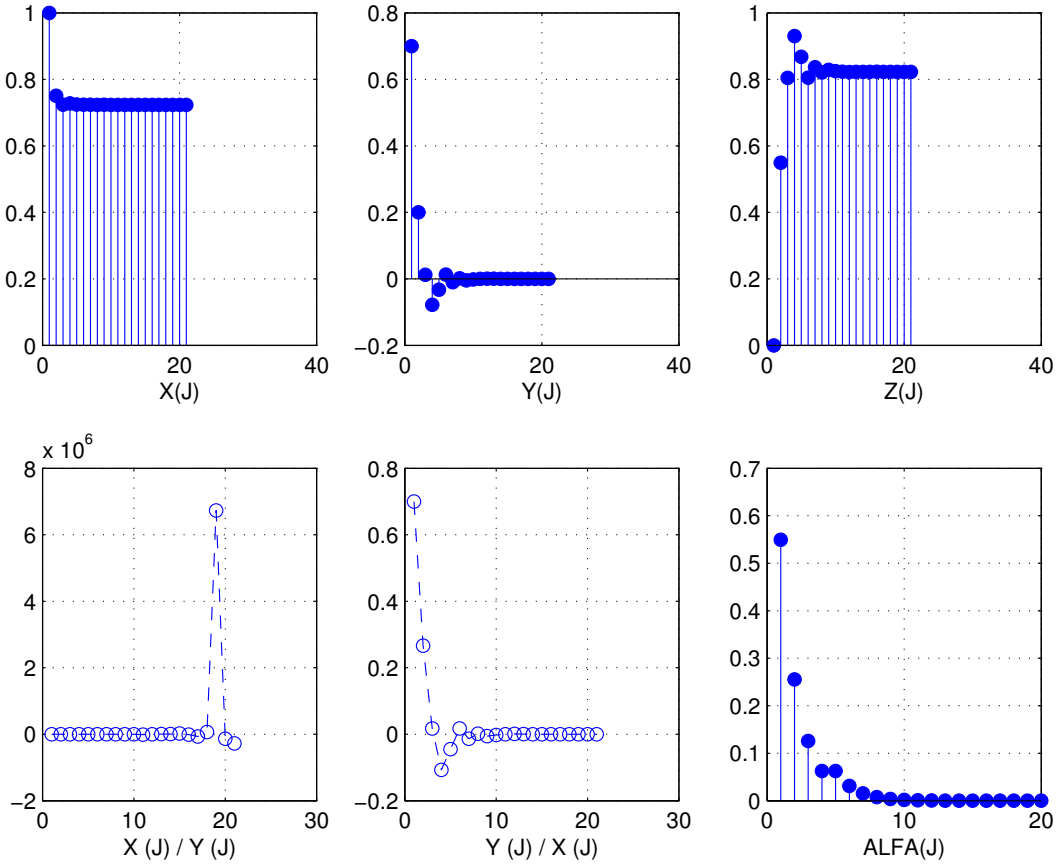

Figura 2.28: Gráfico da função Tangente Inversa Hiperbólica Cordic

2.8.12 Função Cosseno Hiperbólico Cordic ($cosh\_c$)

A função cosseno hiperbólico cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Z-Reduction.

No gráfico a seguir, podemos ver o cosseno hiperbólico cordic de 1 radiano com 20 bits de precisão, como observamos na posição da variável x (já multiplicada pelo fator de correção de escala).

Cosseno Hiperbólico CORDIC	
Comando	cosh_c(1,20)
Coordenadas	Hiperbólico
Modo de Operação	Z-Reduction
Resultado CORDIC	1,5431
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Ângulos $\leq$ 1,1181

Figura 2.29: Cosseno Hiperbólico Cordic

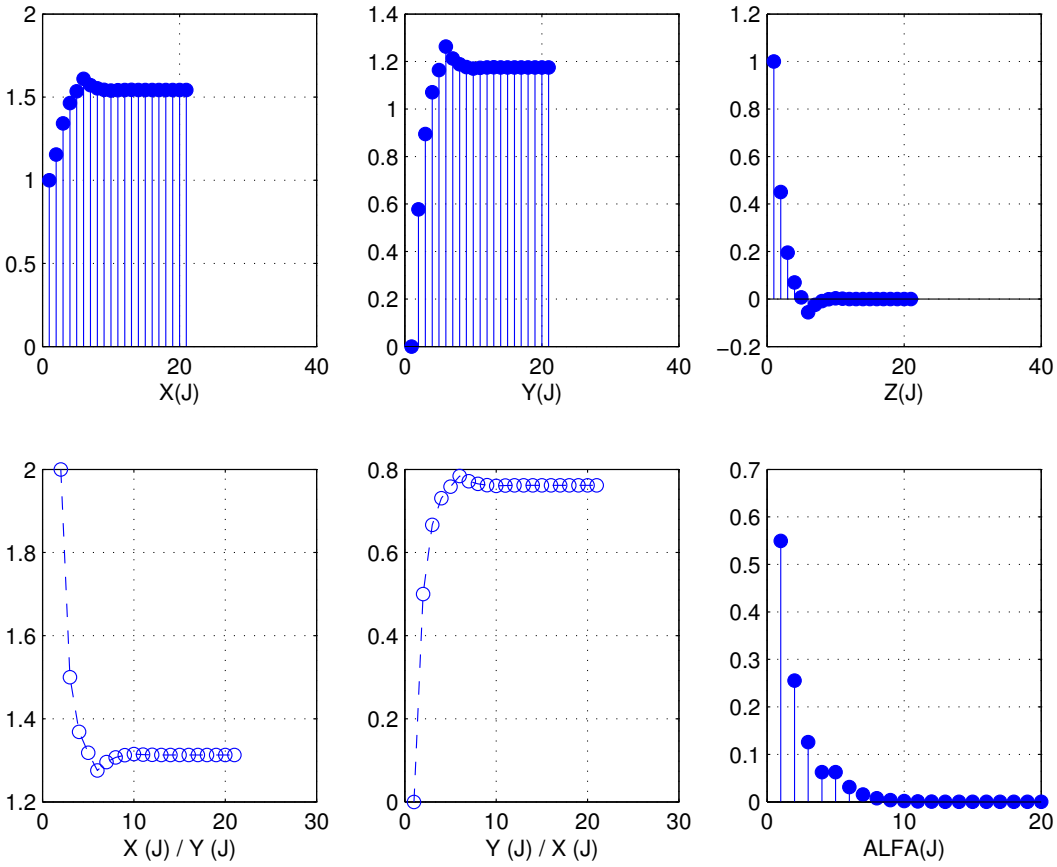

Figura 2.30: Gráfico da função Cosseno Hiperbólico Cordic

2.8.13 Função Seno Hiperbólico Cordic ($sinh\_c$)

A função seno hiperbólico cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Z-Reduction.

Na Figura 2.30 podemos ver o seno hiperbólico cordic de 1 radiano com 20 bits de precisão na posição da variável y.

Seno Hiperbólico CORDIC	
Comando	sinh_c(1,20)
Coordenadas	Hiperbólico
Modo de Operação	Z-Reduction
Resultado CORDIC	1,1752
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Ângulos ≤ 1,1181

Figura 2.31: Seno Hiperbólico Cordic

2.8.14 Função Raiz Quadrada Cordic ($sqrt\_c$)

A função raiz quadrada cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Y-Reduction.

No próximo gráfico, podemos ver a raiz quadrada cordic de 1,2 unidades com 20 bits de precisão, como observamos na posição da variável x (o fator de correção de escala foi multiplicado a cada iteração). As variáveis x e y são iniciadas com os valores $x = \tau + \frac{1}{4}$ e $y = \tau - \frac{1}{4}$, sendo τ o valor desejado para cálculo da raiz (ver Tabela 2.5).

Neste caso, o algoritmo converge para $\tau \leq 2,3816$, devido à inequação $atanh(\frac{y}{x}) \leq 1,1182$, que é o A_0 máximo de convergência dada pela seqüência de deslocamento.

Raiz Quadrada CORDIC	
Comando	sqrt_c(1.2,20)
Coordenadas	Hiperbólico
Modo de Operação	Y-Reduction
Resultado CORDIC	1,0954
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Números ≤ 2,3816

Figura 2.32: Raiz Quadrada Cordic

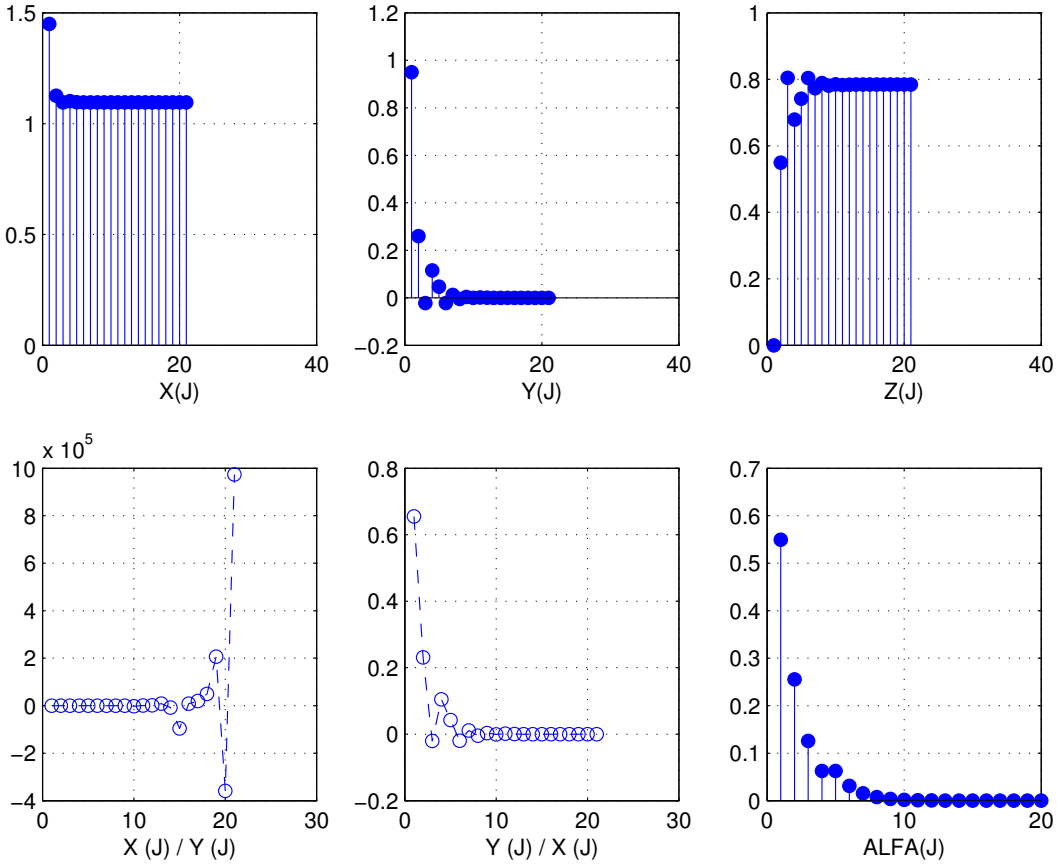

Figura 2.33: Gráfico da função Raiz Quadrada Cordic

2.8.15 Função Exponencial Cordic (exp_c)

A função exponencial cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Z-Reduction.

A função exponencial é dada pela soma do seno hiperbólico com o cosseno hiperbólico, ou seja, pela soma das variáveis x e y. No gráfico abaixo podemos ver a exponencial cordic de 1 unidade com 20 bits de precisão, mostrada na posição $x+y$.

Nesta função, as variáveis x, y e z são iniciadas com os valores $x = 1$, $y = 0$ e $z = \tau$, sendo τ o número desejado para cálculo do exponencial. O algoritmo converge para qualquer número real menor do que 1,1181, devido ao máximo ângulo de rotação.

Exponencial CORDIC	
Comando	exp_c(1,20)
Coordenadas	Hiperbólico
Modo de Operação	Z-Reduction
Resultado CORDIC	2,7183
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Números $\leq$ 1,1181

Figura 2.34: Exponencial Cordic

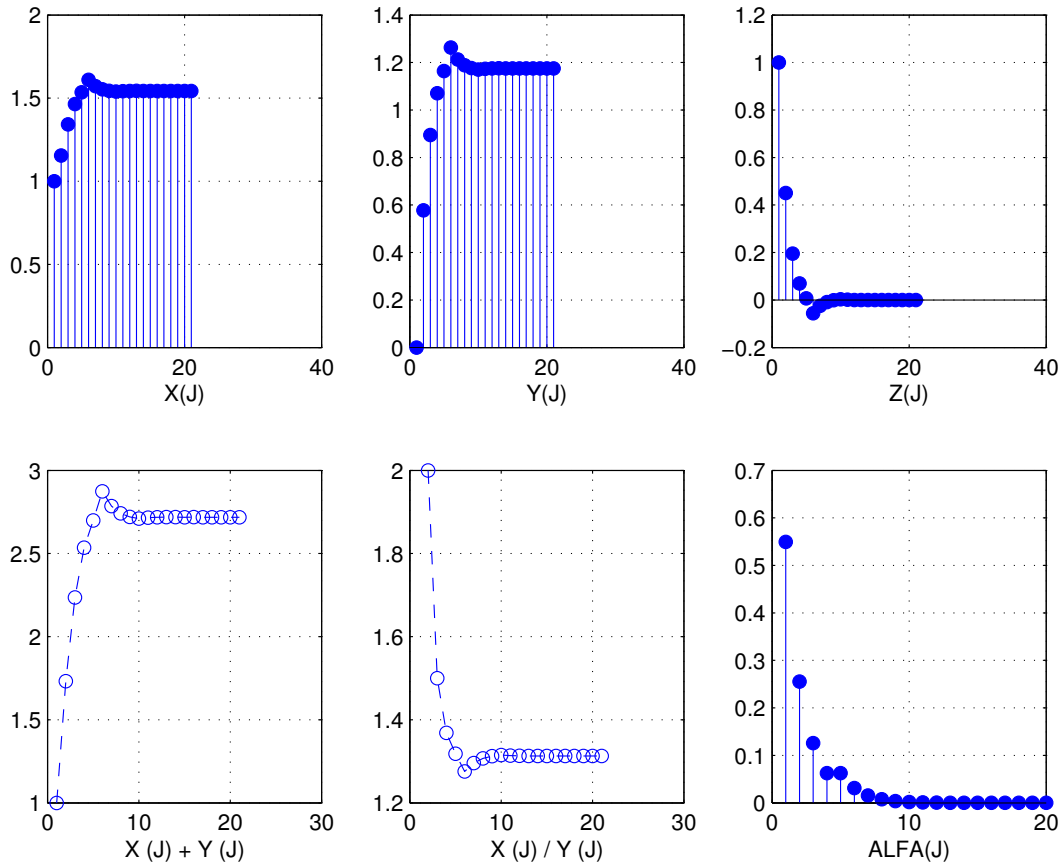

Figura 2.35: Gráfico da função Exponencial Cordic

2.8.16 Função Logaritmo Neperiano Cordic (ln_c)

A função logaritmo neperiano cordic é obtida com o sistema de coordenadas hiperbólico ($m = -1$), no modo de operação Y-Reduction.

No gráfico a seguir podemos ver o logaritmo neperiano cordic de 9 unidades com 20 bits de precisão, na posição da variável z, multiplicada por 2. Isto porque o logaritmo neperiano é igual a $2 \cdot tanh^{-1}(\frac{y}{x})$, sendo $x = z + 1$ e $y = z - 1$ (ver Tabela 2.5).

O algoritmo converge para qualquer número real menor do que 9,3, devido à inequação $atanh(\frac{y}{x}) \leq 1,1182$, que é o A_0 máximo de convergência dada pela seqüência de deslocamento. Neste caso, as variáveis x e y são iniciadas com os valores $x = \tau + 1$ e $y = \tau - 1$, sendo τ o número desejado para cálculo do logaritmo.

Logaritmo Neperiano CORDIC	
Comando	ln_c(9,20)
Coordenadas	Hiperbólico
Modo de Operação	Y-Reduction
Resultado CORDIC	2,1972
Kn	0,8282
$\|A_0\|$	1,1182
Acurácia	20 bits
Convergência	Números $\leq$ 9,3

Figura 2.36: Logaritmo Neperiano Cordic

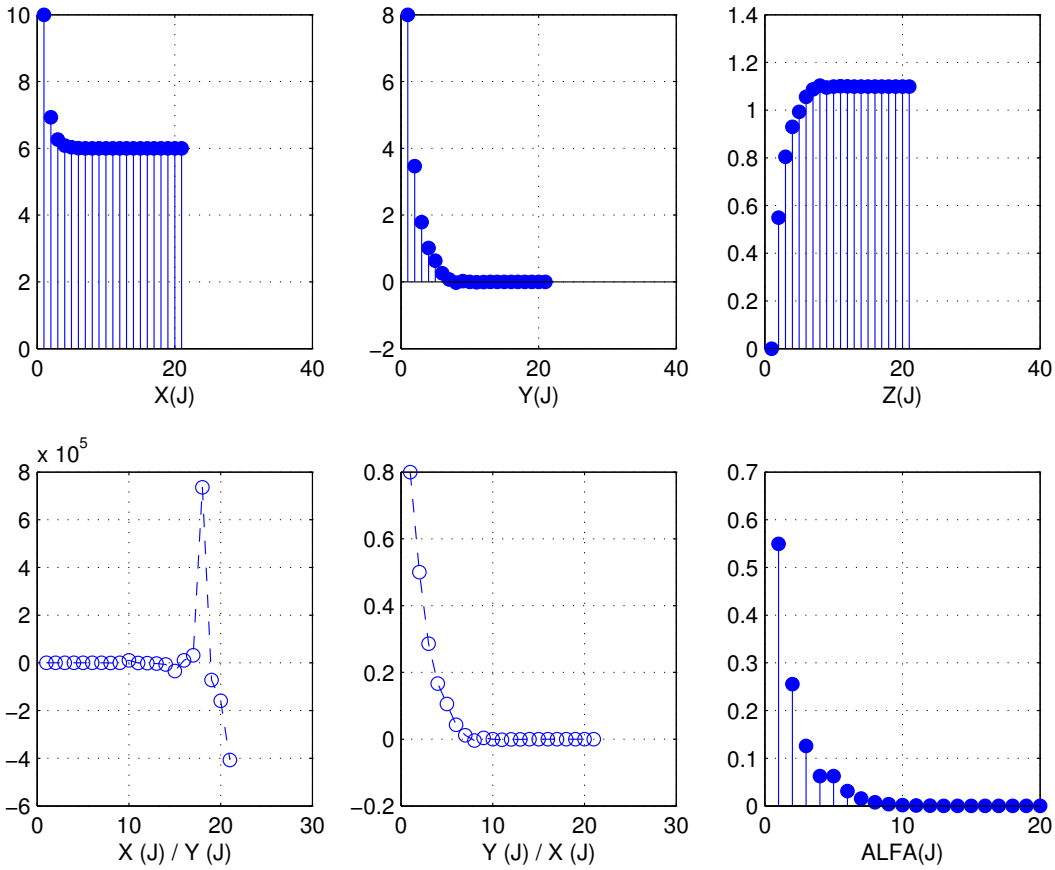

Figura 2.37: Gráfico da função Logaritmo Neperiano Cordic

2.9 Resultados

As implementações das 16 funções foram importantes para o correto entendimento do algoritmo, de suas peculiaridades e de suas imperfeições. Garantir a região de convergência é uma condição necessária para uso do Cordic, sendo a escolha de uma seqüência de deslocamento peça chave nesta solução.

Com uma gama de operações CORDIC à disposição, há diversas maneiras possíveis para prover precisão finita de n bits a uma aplicação desejada.

Capítulo 3

Decomposição em Valores Singulares

3.1 Introdução

A decomposição em valores singulares pode ser aplicada a qualquer matriz $\in C^{m \times n}$, possuindo diversas aplicações nas áreas de processamento de imagens, cálculo e redução do posto de uma matriz, para decomposição em coordenadas polares e mesmo para cálculo dos mínimos quadrados, sendo uma das técnicas mais usadas na decomposição de matrizes ao lado da fatoração LU e da fatoração QR.

3.2 Conceito de SVD

A decomposição dos valores singulares de uma matriz $M \in C^{m \times n}$ é dada por:

$$M = U \cdot \Sigma \cdot V^H \tag{3.1}$$

Sendo $U \in C^{m \times m}$ e $V \in C^{n \times n}$ são matrizes unitárias e $\Sigma \in R^{m \times n}$ é uma matriz diagonal não negativa.

Observando a equação (3.1), vemos a semelhança entre a decomposição SVD e a decomposição espectral (autovalor-autovetor) de uma matriz simétrica $M = Q \cdot \Lambda \cdot Q^T$ [13]. A relação entre elas é analisada da seguinte forma: na decomposição

espectral, os autovalores de M formam a matriz diagonal Λ, pois eles são os seus elementos não nulos. A matriz Q é ortogonal ($Q \cdot Q^T = I$), e os autovetores de uma matriz simétrica podem ser transformados em autovetores ortonormais. No caso da SVD, a matriz diagonal real não negativa Σ é formada pelos seus valores singulares, ordenados em forma decrescente. As matrizes U e V são formadas por vetores ortonormais.

Outra propriedade da SVD é a visualização imediata do posto da matriz decomposta, tornando a SVD muito atrativa dentro da álgebra linear. Isto porque muitas propriedades na álgebra linear só são válidas se a matriz tiver posto completo. O posto de uma matriz qualquer M é igual ao número de valores singulares diferentes de 0. Os valores singulares σ_i são ordenados de forma decrescente tal que:

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq \sigma_r > 0, \qquad \text{onde r = Posto (M)}$$

Para calcular a SVD de uma matriz quadrada ou retangular, o primeiro passo do algoritmo da SVD é multiplicar MM^T e M^TM. As colunas de $U \in C^{m \times m}$ são autovetores de MM^T, e as colunas de $V \in C^{n \times n}$ são autovetores de M^TM. Os r valores singulares na diagonal de $\Sigma \in R^{m \times n}$ são raízes quadradas de autovalores não nulos de MM^T e M^TM. O resultado da decomposição resulta nas matrizes U, Σ e V dispostas como:

$$\text{U} = [\text{u}_1, \text{u}_2 , \ldots , \text{u}_m] \text{ , } \Sigma = \text{diag } [\sigma_1, \sigma_2 , \ldots , \sigma_n] \text{ , e V} = [\text{v}_1, \text{v}_2 , \ldots , \text{v}_m]$$

Outra característica importante da SVD é o fato de algumas normas terem seu cálculo diretamente interligado com os valores singulares, como é o caso da Norma de Frobenius e a norma-2. Pode-se obter o resultado destas normas conhecendo os valores singulares, como está explicitado a seguir:

$$\|M\|_F^2 = \sigma_1^2 + \ldots + \sigma_p^2 \qquad p = min\{m, n\} \tag{3.2}$$

$$\|M\|_2 = \sigma_1 \tag{3.3}$$

A complexidade da SVD está no método de diagonalização da matriz. Uma matriz é diagonalizável se seus autovetores são linearmente independentes. No desenvolvimento desta tese, trabalhamos com o algoritmo de Jacobi para a computação da SVD, um dos primeiros métodos que apareceram na literatura. O método de Jacobi é bem interessante porque ele tem acurácia superior dentro de certas circunstâncias e pode ser utilizado com computação paralela. Além dos métodos de Jacobi para computação da SVD, há ainda os métodos de Golub-Kahan SVD, o Golub-Reinsch SVD e o R-SVD, além da implementação da GSVD [1].

3.3 Métodos de Jacobi

Os métodos de Jacobi realizam uma seqüência de rotações para diagonalizar uma matriz simétrica e quadrada $M \in R^{n \times n}$, com a propriedade de que cada nova matriz M' seja "*mais diagonal*" que a matriz M predecessora ($M' = J^T M J$).

Após as rotações ortogonais, o resultado da norma dos elementos da *não-diagonal* (também chamada de "*off-diagonal*") é pequeno o suficiente para ser desprezada.

$$off(M) = \sqrt{\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} m_{ij}^2} \tag{3.4}$$

A rotação responsável por diagonalizar a matriz simétrica é chamada de rotação de Jacobi. A denominação J (p, q, θ) é dada para uma rotação de Jacobi de ângulo θ no plano (p, q), sendo determinada pelo algoritmo a seguir:

$$\begin{cases} j_{ii} = 1, \qquad \forall (i \neq p, q) \\ j_{pp} = \cos\phi, \qquad j_{pq} = sen\phi \\ j_{qp} = -sen\phi, \qquad j_{qq} = \cos\phi \end{cases} \qquad \text{Para todos outros, } j_{ij} = 0 \tag{3.5}$$

A forma matricial da rotação de Jacobi J pode ser vista abaixo:

$$J(p,q,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos & \cdots & sen & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -sen & \cdots & \cos & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \end{matrix} \qquad (3.6)$$
$$\begin{matrix} & & p & & q & & \end{matrix}$$

Podemos observar que as rotações de Givens não são diferentes das rotações de Jacobi.

Para calcular o autovalor de Jacobi, é necessário escolher um par de índices (p, q) que satisfaça $1 \leq p < q \leq n$, e formar um par de cosseno-seno (cos,sen) tal que:

$$\begin{bmatrix} m_{pp}^{'} & m_{pq}^{'} \\ m_{qp}^{'} & m_{qq}^{'} \end{bmatrix} = \begin{bmatrix} \cos & sen \\ -sen & \cos \end{bmatrix}^T * \begin{bmatrix} m_{pp} & m_{pq} \\ m_{qp} & m_{qq} \end{bmatrix} * \begin{bmatrix} \cos & sen \\ -sen & \cos \end{bmatrix} \qquad (3.7)$$

O método se completa atualizando M sucessivamente da forma:

$$M' = J^T \cdot M \cdot J, \qquad \text{onde } J = J(p,q,\theta) \qquad (3.8)$$

Após cada rotação é necessário verificar se a norma de *Frobenius* foi mantida. A norma de Frobenius de uma matriz qualquer é dada pela fórmula:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2} \qquad (3.9)$$

Portanto, se observarmos a norma de *Frobenius* da matriz $M^{'}$, veremos que é preservada pelas transformações ortogonais feitas em M, tal que:

$$off\,(M')^2 = off\,(M)^2 - 2m_{pq}^2 \tag{3.10}$$

Isto mostra porque o método de Jacobi vai se aproximando da forma diagonal a cada passo de Jacobi.

A resolução da equação (3.7) também é conhecida como Decomposição de Shur, pois determinará a matriz de Givens do par ótimo (p, q).

Antes de entrar no mérito da escolha do par ótimo (p, q), a Decomposição de Schur será abordada de forma mais detalhada.

3.3.1 Decomposição de Schur

O par (cos, sen) na equação (3.7) é determinado por uma decomposição de Schur da matriz simétrica M tal que $M^{'} = J(p,q,\theta)^T \cdot M \cdot J(p,q,\theta)$. O fato de diagonalizar a matriz M faz com que:

$$0 = m_{pq}^{'} = m_{qp}^{'} = m_{pq}\left(\cos^2 - sen^2\right) + (m_{pp} - m_{qq})\cos * sen \tag{3.11}$$

Se $m_{pq} = 0$, então o par (cos, sen) = (1,0). No caso de $m_{pq} \neq 0$, definimos uma variável auxiliar τ igual a:

$$\tau = \frac{m_{qq} - m_{pp}}{2m_{pq}} \tag{3.12}$$

Reescrevendo as equações (3.11) e (3.12) em função de $t = tan(\theta)$, tem-se como resultado uma equação de segundo grau, tal que:

$$t^2 + 2\tau \cdot t - 1 = 0 \tag{3.13}$$

Ou seja, a escolha do par (cos, sen) depende da escolha de t tal que a norma de *Frobenius* da diferença entre $M^{'}$e$M(\|M^{'} - M\|_F))$ seja menor possível.

3.3.2 Método Clássico de Jacobi

Como foi mencionado anteriormente, as rotações de Jacobi apenas alteram as linhas e as colunas p e q. Mas como escolher o par p e q a fim de otimizar a decomposição de Schur?

Pensando conforme a equação (3.10), do ponto de vista da redução da norma dos elementos da *não-diagonal* da matriz M - *off(M)* - faz sentido a escolha do par (p, q) tal que $(m_{pq})^2$ seja máximo. Essa é a idéia do algoritmo clássico de Jacobi.

Algumas vertentes do método clássico de Jacobi, mas com menos complexidade na busca pelo par (p, q) são os métodos "*Cyclic-by-Row*" e "*Cyclic-by-Column*".

3.3.3 Algoritmos Cyclic-by-Row e Cyclic-by-Column

O problema do método Clássico de Jacobi descrito anteriormente é que a atualização da matriz tem complexidade *O(n)* enquanto que a busca por um par ótimo (p, q) envolve $O(n^2)$. Uma maneira de minimizar este problema seria escolher um par qualquer (i, j) através de dois algoritmos: o "*cyclic-by-row*" e o "*cyclic-by-column*". Estes métodos cíclicos de Jacobi convergem quadraticamente, e como eles não requerem busca no cálculo da *não-diagonal*, eles são considerados mais rápidos que o algoritmo de Jacobi original.

O algoritmo *cyclic-by-row* está exemplificado a seguir:

$$
\begin{aligned}
&(i_0, j_0) = (1, 2)\,, \\
&(i_{k+1}, j_{k+1}) = \begin{cases} (i_k, j_{k+1})\,, & \Rightarrow \text{se } i_k < n-1, \quad j_k < n, \\ (i_k + 1, i_k + 2)\,, & \Rightarrow \text{se } i_k < n-1, \quad j_k = n, \\ (1, 2)\,, & \Rightarrow \text{se } i_k = n-1, \quad j_k = n, \end{cases}
\end{aligned}
\tag{3.14}
$$

Exemplificando o caso de $n = 4$, o par escolhido seria:

$$(p, q) = (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), \ldots$$

No caso do algoritmo *cyclic-by-column*, a escolha do par é feita da seguinte maneira:

$$\begin{aligned} &(i_0, j_0) = (1, 2)\,, \\ &(i_{k+1}, j_{k+1}) = \begin{cases} (i_k + 1, j_k)\,, & \Rightarrow \text{se } i_k < j_k - 1, \quad j_k \leq n, \\ (1, j_k + 1)\,, & \Rightarrow \text{se } i_k = j_k - 1, \quad j_k < n, \\ (1, 2)\,, & \Rightarrow \text{se } i_k = n - 1, \quad j_k = n, \end{cases} \end{aligned} \tag{3.15}$$

3.4 Implementação: SVD pelo Método de Jacobi no Matlab

Para implementação da SVD pelo Método de Jacobi, foram criadas 4 funções no Matlab, e a função principal é a função "*svd_j.m*". Esta função executa outras três funções auxiliares e seu código está descrito em função do algoritmo *cyclic-by-row* para determinação do par ótimo (p, q). As outras funções calculam a decomposição de Schur, a matriz de rotação de Jacobi e calculam a *não-diagonal* seguindo passo a passo a teoria de Jacobi elucidada nas sessões anteriores.

As funções implementadas no Matlab para o cálculo da SVD foram:

- **svd_j.m** $\longrightarrow$ Função principal, responsável por ordenar em ordem decrescente os valores singulares da matriz Σ e seus respectivos autovetores nas matrizes unitárias U e V. Tem como valor de entrada uma matriz simétrica real, e retorna as matrizes Σ, U e V.

- **schur2×2.m** $\longrightarrow$ Função que realiza a decomposição de Schur, determinando o cosseno e o seno da rotação de Jacobi a ser aplicada na matriz desejada. Tem como valores de entrada a matriz simétrica e o par (p, q) determinado pelo *cyclic-by-row*. Retorna os valores do seno e cosseno.

- **jacobimatrix.m** $\longrightarrow$ Função que prepara a matriz de rotação de Jacobi *J*. Tem como valores de entrada o cosseno e o seno da matriz de Givens, a matriz simétrica desejada e o par ótimo (p, q). Retorna a matriz de rotação de Jacobi.

- **off.m** $\longrightarrow$ Função que calcula a norma dos elementos da *não-diagonal* na matriz desejada. Tem como valor de entrada a matriz simétrica e retorna o valor da *não-diagonal*.

Como visto na Seção 3.3, o Método de Jacobi necessariamente tem que ser aplicado a matrizes simétricas e quadradas. É feito um teste na função "**svd_j.m**" para verificar esta condição. Se a matriz de entrada não for simétrica, a função retorna um texto contendo a mensagem: *ERRO!!! A Matriz tem que ser SIMÉTRICA*.

Para comprovar o funcionamento da implementação, a função "*svd_j.m*" é comparada com a função *svd.m*, proprietária do Matlab. O comando help [função] também pode ser usado, facilitando a simulação. A seguir estão os resultados da implementação da SVD pelo método de Jacobi de matrizes simétricas 3×3 e 5×5.

3.4.1 Exemplo com matriz 3×3

$$B = \begin{bmatrix} -1 & -2 & -3 \\ -2 & -4 & -2 \\ -3 & -2 & -8 \end{bmatrix} \tag{3.16}$$

Ao decompor a matriz B em seus valores singulares usando a "*svd_j.m*", tem-se:

$$\begin{bmatrix} -1 & -2 & -3 \\ -2 & -4 & -2 \\ -3 & -2 & -8 \end{bmatrix} = \begin{bmatrix} 0,3610 & 0,1518 & -0,9201 \\ 0,3898 & 0,8718 & 0,2968 \\ 0,8472 & -0,4658 & 0,2556 \end{bmatrix} \cdot \tag{3.17}$$

$$\cdot \begin{bmatrix} 10,1988 & 0 & 0 \\ 0 & 3,2796 & 0 \\ 0 & 0 & 0,4784 \end{bmatrix} \cdot \begin{bmatrix} -0,3610 & -0,1518 & -0,9201 \\ -0,3898 & -0,8718 & 0,2968 \\ -0,8472 & 0,4658 & 0,2556 \end{bmatrix}^T \tag{3.18}$$

O resultado da decomposição pode ser comprovado de duas maneiras, tanto no cálculo da função *svd.m* proprietária do Matlab (que não decompõe a matriz pelo método de Jacobi), quanto pela prova natural da decomposição, que é a multiplicação das matrizes $U \cdot \Sigma \cdot V$, e resulta exatamente na matriz A original.

3.4.2 Exemplo com matriz 5×5

$$C = \begin{bmatrix} 1 & 2 & -3 & 4 & 5 \\ 2 & 3 & 10 & 13 & 4 \\ -3 & 10 & 9 & -10 & 3 \\ 4 & 13 & -10 & 27 & 2 \\ 5 & 4 & 3 & 2 & 81 \end{bmatrix} \tag{3.19}$$

Ao decompor a matriz B em seus valores singulares usando a "*svd_j.m*", tem-se:

$$\begin{bmatrix} 1 & 2 & -3 & 4 & 5 \\ 2 & 3 & 10 & 13 & 4 \\ -3 & 10 & 9 & -10 & 3 \\ 4 & 13 & -10 & 27 & 2 \\ 5 & 4 & 3 & 2 & 81 \end{bmatrix} = \begin{bmatrix} -0,0640 & -0,1398 & 0,0865 & 0,1289 & -0,9758 \\ -0,0653 & -0,2906 & -0,6056 & -0,7305 & -0,1043 \\ -0,0405 & 0,2647 & -0,7868 & 0,5552 & -0,0316 \\ -0,0490 & -0,9067 & -0,0443 & 0,3762 & 0,1789 \\ -0,9938 & 0,0620 & 0,0685 & -0,0015 & 0,0622 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} 81,8061 & 0 & 0 & 0 & 0 \\ 0 & 34,5651 & 0 & 0 & 0 \\ 0 & 0 & 16,2034 & 0 & 0 \\ 0 & 0 & 0 & 11,6391 & 0 \\ 0 & 0 & 0 & 0 & 0,0646 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} -0,0640 & -0,1398 & 0,0865 & -0,1289 & -0,9758 \\ -0,0653 & -0,2906 & -0,6056 & 0,7305 & -0,1043 \\ -0,0405 & 0,2647 & -0,7868 & -0,5552 & -0,0316 \\ -0,0490 & -0,9067 & -0,0443 & -0,3762 & 0,1789 \\ -0,9938 & 0,0620 & 0,0685 & 0,0015 & 0,0622 \end{bmatrix}^T$$

De forma análoga, a decomposição acima é idêntica à decomposição dos valores singulares pela função *svd.m* do Matlab.

3.4.3 Resultados

A implementação da SVD pelo método de Jacobi apresentou os mesmos resultados obtidos pela função genérica SVD proprietária do Matlab, conforme o esperado.

A acurácia da implementação *svd_j.m* ficou restrita ao comprimento da palavra do Matlab, que limita os erros de aproximação com a resolução de 10^{-14} para variáveis do tipo "*long*".

Apesar do algoritmo de Jacobi ter suas limitações (somente trabalhar com matrizes simétricas e quadradas reais) ele foi escolhido pelas suas vantagens de paralelismo e complexidade, para que dentro de seu código fonte seja feita a adaptação necessária para introduzir as funções CORDIC. No capítulo a seguir, será mostrada

uma idéia de como pode ser feita esta integração, visando suas vantagens de acurácia e redução de posto.

Capítulo 4

SVD com precisão CORDIC

4.1 Implementação: Algoritmo SVD_CORDIC no Matlab

Uma vez implementada a SVD pelo método de Jacobi e 16 funções CORDIC capazes de fazer operações com acurácia desejada, será implementado neste capítulo um algoritmo capaz de mesclar técnicas de redução do posto com as vantagens de uma "calculadora elegante de precisão variável".

A princípio, pode parecer que a construção de um algoritmo *svd_cordic* seja intuitiva e instantânea. Porém, se todas as operações existentes no código da função *svd_j.m* fossem substituídas por operações CORDIC, surgiria a necessidade de analisar a região de convergência em cada operação CORDIC, além de possíveis erros de arredondamento que não poderiam ser desprezados. Este não é o objetivo da tese, pois para obter-se um método de decompor matrizes com precisão e posto reduzidos, só é preciso controlar a acurácia do algoritmo. Isto pode ser feito de diversas maneiras dentro da implementação de Jacobi, desde que sejam conhecidas todas as operações que compõem o código fonte.

Na implementação da SVD (Capítulo 3), além das quatro operações fundamentais da matemática, foram usadas algumas funções proprietárias do Matlab no intuito de simplificar o código final (como por exemplo, as funções *transpose*, *size* e *for*). A Tabela 4.1 ilustra quantas operações aritméticas, trigonométricas e funções do Matlab foram usadas em cada função do método de Jacobi, independente de ser

operações entre vetores ou entre matrizes.

Tabela 4.1: Número de operações aritméticas, trigonométricas e funções do Matlab que compõem cada função da SVD_J

SVD_JACOBI					
Tabela de Operações	**svd_j**	**off**	**schur2x2**	**jacobimatrix**	**Total**
+	6	1	5		12
-	6		1		7
*	11		4	1	16
/			4		4
^		1	3		4
Funções SVD_JACOBI					
off	2	-			2
schur2x2	1		-		1
jacobimatrix	1			-	1
Funções Matlab					
size	1	1	1	1	4
for	6	2			8
if	3	1	2		6
fprintf	1				1
eye	1			1	2
while	1				1
transpose	3				3
abs	1				1
max	2				2
wshift	2				2
wkeep	1				1
return	1				1
sqrt		1	3		4

Para trabalhar com uma precisão de n bits, é necessário substituir algumas operações por funções CORDIC, observando a região de convergência desta operação. A decomposição de Schur foi escolhida para fazer esta substituição, pois é a função que retorna o *seno* e o *cosseno* da matriz de rotação de Jacobi. Este passo é a alma do algoritmo de Jacobi, pois a cada iteração a norma da *não-diagonal* vai diminuindo e, por conseqüência, a matriz vai se tornando mais diagonal.

Ao observar a coluna *schur2×2* da Tabela 4.1, veremos que esta função é composta de 23 operações, das quais destacam-se as operações raiz quadrada, multiplicação, divisão e potenciação que definem no código fonte o resultado do *seno* e *cosseno* (ver equação (3.13)). Substituindo estas quatro operações pelas funções *sqrt_c*, *mult_c* e *div_c*, implementamos a decomposição de uma matriz em seus valores singulares pelo método de Jacobi, com uma precisão em bits variável. Esta

alteração originou o algoritmo *svd_cordic*.

Foram criadas oito funções no Matlab, sendo a função principal denominada *svd_c.m*. Cada uma delas está descrita a abaixo:

- **svd_c.m** $\longrightarrow$ Função principal, responsável por ordenar em ordem decrescente os valores singulares da matriz Σ e seus respectivos autovetores nas matrizes unitárias U e V. Tem como valor de entrada uma matriz simétrica real e os bits de precisão do CORDIC, e retorna as matrizes Σ, U e V com a acurácia desejada.

- **schur2×2_cordic.m** $\longrightarrow$ Função que realiza a decomposição de Schur com determinada acurácia. Sua diferença com relação à função *schur2×2.m* está no cálculo do seno e cosseno da rotação de Jacobi, onde as operações de multiplicação e raiz quadrada usadas na decomposição são feitas pelo CORDIC, com a acurácia determinada pelo número de bits informado em *svd_c*. Tem como valores de entrada a matriz simétrica, o par (p, q) determinado pelo *cyclic-by-row* e os bits de precisão, retornando os valores do seno e cosseno "arredondados".

- **jacobimatrix.m** $\longrightarrow$ Mesma função usada na implantação da *svd_j*, responsável por calcular a matriz de rotação de Jacobi *J*. Tem como valores de entrada o cosseno e o seno da matriz de Givens, a matriz simétrica desejada e o par ótimo (p, q). Retorna a matriz de rotação de Jacobi.

- **off.m** $\longrightarrow$ Também é a mesma função usada na implementação *svd_j* para calcular a norma dos elementos da *não-diagonal* na matriz desejada. Tem como valor de entrada a matriz simétrica e retorna o valor da *não-diagonal*.

- **bsign.m** $\longrightarrow$ Função de controle do algoritmo CORDIC. Tem como valor de entrada um número real e retorna 1 se o número é maior ou igual a zero, e retorna -1 se o número é menor que zero (ver Seção 2.8).

- **mult_c.m** $\longrightarrow$ Função que faz a multiplicação CORDIC no sistema de coordenadas linear, modo de operação Z-reduction (ver Tabela 2.5). Tem como valor de entrada os dois números que serão multiplicados e o total de bits de precisão (acurácia desejada). Retorna o valor da multiplicação CORDIC.

- **div_c.m** $\longrightarrow$ Função que faz a divisão CORDIC no sistema de coordenadas linear, modo de operação Y-reduction (ver Tabela 2.5). Tem como valor de entrada os dois números que serão divididos e o total de bits de precisão (acurácia desejada). Retorna o valor da divisão CORDIC.

- **sqrt_c.m** $\longrightarrow$ Função que faz a raiz quadrada CORDIC no sistema de coordenadas hiperbólicas, modo de operação Y-reduction (ver Tabela 2.5). Tem como valor de entrada um número maior ou igual a zero e o total de bits de precisão (acurácia desejada), retornando o valor da raiz quadrada CORDIC.

Para um melhor entendimento do código implementado, a Tabela 4.2 ilustra quantas operações aritméticas, trigonométricas e funções do Matlab foram usadas em cada função do método SVD_CORDIC, independente de ser operações entre vetores ou entre matrizes.

Tabela 4.2: Número de operações aritméticas, trigonométricas e funções do Matlab que compõem cada função da SVD_C

SVD_CORDIC									
Tabela de Operações	**svd_c**	**off**	**schur2x2_cordic**	**jacobimatrix**	**mult_c**	**div_c**	**sqrt_c**	**bsign**	**Total**
+	6	1	6		9	10	11		43
-	6		1		4	2	5		18
*	11		3	1	7	7	21		50
/			4			1	4		9
^		1	4		2	2	4		13
Funções SVD_CORDIC									
off	2	-							2
schur2x2_cordic	1		-						1
jacobimatrix	1			-					1
mult_c			1		-				1
div_c			1			-			
sqrt_c			1				-		1
bsign					2	2	6	-	10
Funções Matlab									
size	1	1	1	1					4
for	6	2			1	1	1		11
if	3	1	3				1	1	9
fprintf	1		1		1	1	1		5
eye	1			1					2
while	1								1
transpose	3								3
abs	1								1
max	2								2
wshift	2								2
wkeep	1								1
return	1								1
sqrt		1	2		1	1	4		9
break			1						1
sum					1	1	1		3
sign								1	1

A escolha da precisão em bits vai determinar o número de rotações CORDIC a serem realizadas nas funções *mult_c*, *div_c* e *sqrt_c*, sendo o número de operações final diretamente proporcional à ordem da matriz e à acurácia do algoritmo.

4.1.1 Região de Convergência da SVD_CORDIC

A região de convergência do algoritmo *svd_cordic* é dada em função de *mult_c*, *sqrt_c* e *div_c* utilizadas na decomposição de Schur.

Foi visto no Capítulo 2 que a função *sqrt_c* converge para números menores que 2,3816, condição que sempre é respeitada na decomposição de Schur na resolução das raízes da equação (3.13).

Quanto à função *mult_c*, vimos também que converge quando o segundo número y for menor ou igual a 2. Como o cosseno é uma função com limites em $\pm$ 1, a convergência é atingida independente do valor da tangente do ângulo de rotação.

A função *div_c* converge quando $y \leq x \cdot 2$. Pelos mesmos motivos apresentados para a função *sqrt_c*, a convergência é sempre respeitada.

No final da função *schur2×2_cordic* é realizado um teste rápido de convergência, como uma "verificação" do funcionamento do algoritmo. Em caso de erro, é retornado um texto com a mensagem: *ERRO de CONVERGÊNCIA*.

4.2 Testes com Matrizes Simétricas 2×2

Para verificar o funcionamento da implementação, a função *"svd_c.m"* é comparada com a função *svd.m*, proprietária do Matlab. O comando help [função] também pode ser usado à vontade, facilitando a simulação.

Supondo a matriz simétrica :

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \tag{4.1}$$

Decompondo A em seus valores singulares, temos $\sigma_1 = 4,2361$ e $\sigma_2 = 0,2361$. Calculando a SVD da matriz A usando a *svd_c.m*, com precisão variando de 1 a 20 bits, pode ser traçado o seguinte gráfico para cada valor singular:

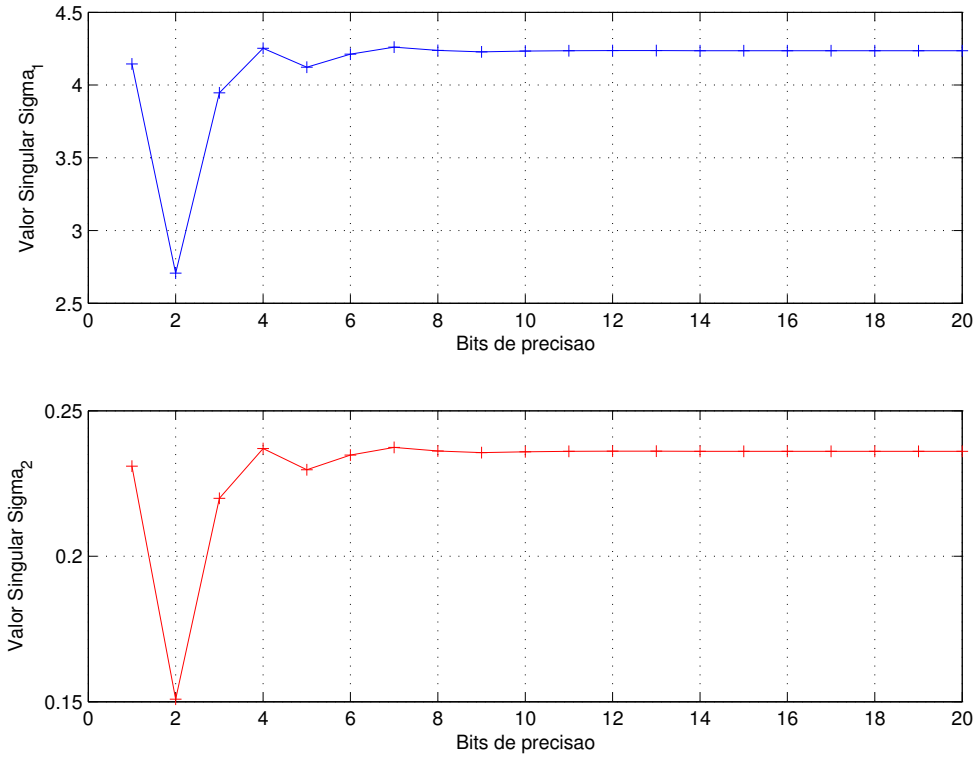

Figura 4.1: Precisão dos valores singulares variando de 1 a 20 bits

A Figura 4.1 mostra que a partir de 10 bits, a acurácia do algoritmo se encontra na quarta casa decimal, fornecendo um resultado satisfatório com relação ao resultado desejado e calculado pelo Matlab. Observa-se também na figura acima que o resultado com 2 bits de precisão apresentou um ponto fora da curva. Lembrando que o número de bits de precisão é igual ao número de iterações CORDIC, vemos o efeito de uma microrotação "indesejada", pois a primeira iteração já tinha levado os valores singulares a se aproximar do resultado correto.

O resultado da decomposição pode ser comprovado pela função *svd.m* proprietária do Matlab, que apresenta o mesmo valor para uma precisão superior a 10 bits, com o algoritmo significativo sendo a quarta casa decimal.

A seguir, a decomposição será feita passo a passo para a precisão de 3, 5, 7, 10 e 15 bits.

4.2.1 SVD com precisão de 3 bits

$$\begin{bmatrix} 0,8679 & 1,7357 \\ 1,7357 & 2,6036 \end{bmatrix} = \begin{bmatrix} -0,5074 & 0,8210 \\ -0,8210 & -0,5074 \end{bmatrix} \cdot \begin{bmatrix} 3,9463 & 0,0000 \\ 0,0000 & 0,2199 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5074 & -0,8210 \\ -0,8210 & 0,5074 \end{bmatrix}^T \tag{4.2}$$

4.2.2 SVD com precisão de 5 bits

$$\begin{bmatrix} 0,9470 & 1,8941 \\ 1,8941 & 2,8411 \end{bmatrix} = \begin{bmatrix} -0,5186 & 0,8392 \\ -0,8392 & -0,5186 \end{bmatrix} \cdot \begin{bmatrix} 4,1224 & 0,0000 \\ 0,0000 & 0,2297 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5186 & -0,8392 \\ -0,8392 & 0,5186 \end{bmatrix}^T \tag{4.3}$$

4.2.3 SVD com precisão de 7 bits

$$\begin{bmatrix} 1,0114 & 2,0228 \\ 2,0228 & 3,0342 \end{bmatrix} = \begin{bmatrix} -0,5272 & 0,8531 \\ -0,8531 & -0,5272 \end{bmatrix} \cdot \begin{bmatrix} 4,2602 & 0,0000 \\ 0,0000 & 0,2374 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5272 & -0,8531 \\ -0,8531 & 0,5272 \end{bmatrix}^T \tag{4.4}$$

4.2.4 SVD com precisão de 10 bits

$$\begin{bmatrix} 0,9986 & 1,9973 \\ 1,9973 & 2,9959 \end{bmatrix} = \begin{bmatrix} -0,5256 & 0,8504 \\ -0,8504 & -0,5256 \end{bmatrix} \cdot \begin{bmatrix} 4,2332 & 0,0000 \\ 0,0000 & 0,2359 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5256 & -0,8504 \\ -0,8504 & 0,5256 \end{bmatrix}^T \tag{4.5}$$

4.2.5 SVD com precisão de 15 bits

$$\begin{bmatrix} 1,0000 & 2,0000 \\ 2,0000 & 3,0000 \end{bmatrix} = \begin{bmatrix} -0,5257 & 0,8506 \\ -0,8506 & -0,5257 \end{bmatrix} \cdot \begin{bmatrix} 4,2361 & 0,0000 \\ 0,0000 & 0,2361 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5257 & -0,8506 \\ -0,8506 & 0,5257 \end{bmatrix}^T \tag{4.6}$$

4.3 SVD_CORDIC com posto reduzido

Com a função SVD_CORDIC implementada, propomos dois métodos diferentes para reduzir o posto desejado:

1. Reduzir o posto da matriz após o resultado da SVD_CORDIC;
2. Adaptar a SVD_CORDIC para reduzir o posto e aplicar a precisão desejada ao mesmo tempo;

4.3.1 Redução do posto após a quantização CORDIC

A redução de posto é feita da seguinte maneira:

- É escolhida a precisão desejada em "bits" para a decomposição nos valores singulares;

- Uma matriz A de dimensões $n \times n$ é decomposta pela função SVD_CORDIC, originando as matrizes $U \cdot \Sigma \cdot V^H$;
- São mantidos os "r" maiores valores singulares da matriz Σ enquanto os demais são zerados, a fim de reduzir o posto desejado.

A figura abaixo ilustra este método:

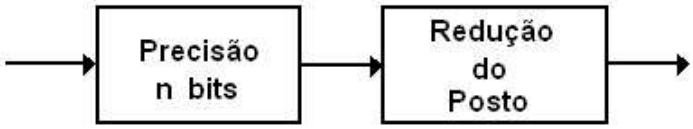

Figura 4.2: Redução do posto após a quantização CORDIC

Se fosse purgado o menor valor singular da matriz A no exemplo acima com precisão de 10 bits, teríamos:

$$\begin{bmatrix} 1,1692 & 1,8919 \\ 1,8919 & 3,0611 \end{bmatrix} = \begin{bmatrix} -0,5256 & 0,8504 \\ -0,8504 & -0,5256 \end{bmatrix} \cdot \begin{bmatrix} 4,2332 & 0,0000 \\ 0,0000 & 0,0000 \end{bmatrix} \cdot \cdot \begin{bmatrix} -0,5256 & -0,8504 \\ -0,8504 & 0,5256 \end{bmatrix}^T \tag{4.7}$$

Apesar do valor singular retirado ser da ordem de 5% do maior valor singular, temos um resultado próximo ao esperado. A seguir vemos um outro exemplo com uma matriz 4×4, sendo decomposta com 10 bits de precisão:

$$A = \begin{bmatrix} 1 & 4 & 7 & 9 \\ 4 & 2 & 8 & 0 \\ 7 & 8 & 3 & 6 \\ 9 & 0 & 6 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 0,9962 & 3,9840 & 6,9875 & 8,9751 \\ 3,9840 & 1,9937 & 7,9880 & 0,0020 \\ 6,9875 & 7,9880 & 2,9989 & 5,9971 \\ 8,9751 & 0,0020 & 5,9971 & 4,9929 \end{bmatrix} = \begin{bmatrix} -0,5239 & 0,5328 & -0,6341 & -0,1818 \\ -0,3598 & -0,5234 & -0,3441 & 0,6901 \\ -0,5620 & 0,3571 & 0,6764 & 0,3151 \\ -0,5285 & -0,5558 & 0,1453 & -0,6246 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} 20,2909 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 7,5441 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 6,3456 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 4,5897 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} -0,5239 & -0,5328 & 0,6341 & -0,1818 \\ -0,3598 & 0,5234 & 0,3441 & 0,6901 \\ -0,5620 & -0,3571 & -0,6764 & 0,3151 \\ -0,5285 & 0,5558 & -0,1453 & -0,6246 \end{bmatrix}^T$$

O menor valor singular é da ordem de 20% do maior valor singular. Se fosse expurgado metade de seus valores singulares, teríamos:

$$\begin{bmatrix} 3,3963 & 5,9445 & 4,5285 & 7,8694 \\ 5,9445 & 0,5595 & 5,5129 & 1,6631 \\ 4,5285 & 5,5129 & 5,4465 & 7,5240 \\ 7,8693 & 1,6631 & 7,5240 & 3,3362 \end{bmatrix} = \begin{bmatrix} -0,5239 & 0,5328 & -0,6341 & -0,1818 \\ -0,3598 & -0,5234 & -0,3441 & 0,6901 \\ -0,5620 & 0,3571 & 0,6764 & 0,3151 \\ -0,5285 & -0,5558 & 0,1453 & -0,6246 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} 20,2909 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 7,5441 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 0,0000 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} -0,5239 & -0,5328 & 0,6341 & -0,1818 \\ -0,3598 & 0,5234 & 0,3441 & 0,6901 \\ -0,5620 & -0,3571 & -0,6764 & 0,3151 \\ -0,5285 & 0,5558 & -0,1453 & -0,6246 \end{bmatrix}^T$$

Neste caso, podemos observar que a quantidade de informação retirada na redução de posto alterou significativamente a matriz **A** original.

4.3.2 Redução do posto em conjunto com a quantização CORDIC

Para reduzir o posto e aplicar a precisão desejada ao mesmo tempo é necessário adaptar a função SVD_CORDIC. Esta adaptação no Matlab deu origem à função SVD_CORDICRED.

O método de Jacobi forma a cada par de cosseno e seno uma nova matriz de rotação, como vimos na Seção 3.3. Vimos também que a SVD_CORDIC substitui as operações da matriz de Givens por operações CORDIC com a precisão desejada. Neste caso, sugerimos que o primeiro ciclo do algoritmo *cyclic-by-row* seja realizado com 1 bit de precisão, para que o posto seja reduzido. Depois de expurgar os menores valores singulares, a precisão é aumentada gradativamente até atingir o patamar desejado.

A precisão e redução de posto ao mesmo tempo são sintetizadas da seguinte forma:

- É escolhida a precisão desejada em "bits" para a decomposição nos valores singulares;
- Uma matriz A de dimensões $n \times n$ é multiplicada por sucessivas matrizes de rotação de Jacobi calculadas com 1 bit de precisão;
- Após o término do *cyclic-by-row* são purgados na diagonal da nova matriz A os menores valores singulares, reduzindo o posto desejável;
- A nova matriz A com posto reduzido (mais diagonal que sua predecessora) é multiplicada por um novo ciclo de rotação de Jacobi calculado com "n" bits de precisão;

A figura abaixo ilustra o método proposto:

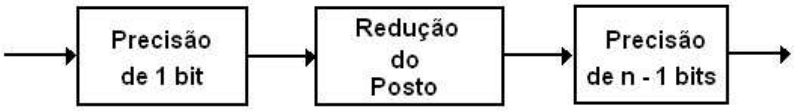

Figura 4.3: Redução do posto em conjunto com a quantização CORDIC

No exemplo acima, fazendo a decomposição com a SVD_CORDICRED passo a passo, tem-se:

- Calculando SVD com 1 bit de precisão:

$$\begin{bmatrix} 0,6208 & 4,8605 & 7,6901 & 7,5771 \\ 4,8605 & 3,3821 & 8,2009 & 0,7401 \\ 7,6901 & 8,2009 & 2,2708 & 3,8528 \\ 7,5771 & 0,7401 & 3,8528 & 2,6164 \end{bmatrix} = \begin{bmatrix} -0,5575 & 0,8713 & -0,1175 & -0,2037 \\ -0,4683 & -0,1269 & -0,6170 & 0,6508 \\ -0,5387 & -0,2452 & 0,7040 & 0,2688 \\ -0,4031 & -0,4628 & -0,2395 & -0,5849 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} 19,4115 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 7,2317 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 6,4954 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 4,0456 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} -0,5575 & -0,8713 & 0,1175 & -0,2037 \\ -0,4683 & 0,1269 & 0,6170 & 0,6508 \\ -0,5387 & 0,2452 & -0,7040 & 0,2688 \\ -0,4031 & 0,4628 & 0,2395 & -0,5849 \end{bmatrix}^T$$

- Reduzindo a matriz dos valores singulares para posto 2:

$$\begin{bmatrix} 0,5426 & 5,8675 & 7,3744 & 7,2779 \\ 5,8675 & 4,1411 & 4,6720 & 3,2396 \\ 7,3744 & 4,6720 & 5,1976 & 3,3939 \\ 7,2779 & 3,2396 & 3,3939 & 1,6048 \end{bmatrix} = \begin{bmatrix} -0,5575 & 0,8713 & -0,1175 & -0,2037 \\ -0,4683 & -0,1269 & -0,6170 & 0,6508 \\ -0,5387 & -0,2452 & 0,7040 & 0,2688 \\ -0,4031 & -0,4628 & -0,2395 & -0,5849 \end{bmatrix} \cdot$$
$$\cdot \begin{bmatrix} 19,4115 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 7,2317 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 0,0000 \end{bmatrix} \cdot$$
$$\cdot \begin{bmatrix} -0,5575 & -0,8713 & 0,1175 & -0,2037 \\ -0,4683 & 0,1269 & 0,6170 & 0,6508 \\ -0,5387 & 0,2452 & -0,7040 & 0,2688 \\ -0,4031 & 0,4628 & 0,2395 & -0,5849 \end{bmatrix}^T$$

- Recalculando a SVD com o restante dos bits de precisão (9 bits):

$$\begin{bmatrix} -0,2200 & 4,3429 & 3,2969 & 7,3942 \\ 4,3424 & 1,9734 & 4,3660 & 0,9262 \\ 3,2965 & 4,3655 & 5,5840 & 5,5056 \\ 7,3964 & 0,9279 & 5,5058 & -2,4979 \end{bmatrix} = \begin{bmatrix} -0,4783 & 0,6040 & -0,2675 & -0,5749 \\ -0,4155 & -0,2401 & 0,8258 & -0,2895 \\ -0,6378 & 0,1999 & -0,0029 & 0,7435 \\ -0,4341 & -0,7320 & -0,4944 & -0,1777 \end{bmatrix}$$
$$\cdot \begin{bmatrix} 14,6942 & 0,0000 & 0,0011 & 0,0014 \\ 0,0016 & 9,8233 & 0,0049 & 0,0011 \\ 0,0003 & 0,0049 & 0,0000 & 0,0014 \\ 0,0016 & 0,0003 & 0,0000 & 0,0000 \end{bmatrix} \cdot$$
$$\cdot \begin{bmatrix} -0,4783 & -0,6040 & 0,2675 & -0,5749 \\ -0,4155 & 0,2401 & -0,8258 & -0,2895 \\ -0,6378 & -0,1999 & 0,0029 & 0,7435 \\ -0,4341 & 0,7320 & 0,4944 & -0,1777 \end{bmatrix}^T$$

4.4 Resultados da SVD_CORDIC

Os resultados obtidos com a SVD_CORDIC e com a SVD_CORDICRED mostram o funcionamento correto do algoritmo com precisão variável.

O Matlab possui para a variável *long* uma resolução de 10^{-14}, e dependendo da precisão desejada e das microrotações fixas CORDIC, o resultado com poucos bits pode ser muito útil para minimizar o número de operações realizadas pelo algoritmo.

A complexidade assintótica do algoritmo SVD_CORDIC é da ordem $O(n^3)$, mesma complexidade assintótica da SVD do Matlab, enquanto que a complexidade assintótica do algoritmo SVD_CORDICRED é da ordem $O(nr^2)$, devido a redução do posto. A explicação é dada pela operação dominante, que é a multiplicação das matrizes decompostas pelos valores singulares, sendo:

$$\mathbf{R}_{yy}^{n\times n} = \mathbf{U}_{n\times n} \cdot \Sigma_{n\times n} \cdot \mathbf{V}_{n\times n}^{H} \tag{4.8}$$

Neste caso, a decomposição tem uma complexidade assintótica de $O(n^3)$. Reduzindo o posto de n para r, temos:

$$\mathbf{R}_{yy}^{n\times n} = \mathbf{U}_{n\times r} \cdot \Sigma_{r\times r} \cdot \mathbf{V}_{r\times n}^{H} \tag{4.9}$$

Após a redução de posto, a operação dominante passa a ter uma complexidade assintótica da ordem de $O(nr^2)$.

Outro ponto relevante é que o número de flops (*Floating Point Operation per Second*) aumenta gradativamente (também da ordem $O(n^3)$) com o número de bits de precisão do algoritmo.

Para entender completamente os resultados dos métodos propostos com precisão variável, estes serão aplicados em um filtro de Wiener.

Capítulo 5

Filtro de Wiener com Posto e Precisão reduzidos

5.1 Introdução

Ao longo dos Capítulos 2, 3 e 4 foi desenvolvida uma ferramenta para decomposição de matrizes nos valores singulares com precisão variável que pode ser usada em diversas aplicações tanto nas áreas de processamento de imagens quanto na álgebra linear.

Para ilustrar o uso desta ferramenta, foi escolhida como aplicação um filtro de Wiener para redução do posto e da precisão. O algoritmo implementado no Matlab decompõe a matriz de autocorrelação que compõe o filtro de Wiener pela SVD_CORDIC. O fruto desta decomposição determina como um filtro de Wiener pode ser quantizado em uma implementação com precisão variável e ao mesmo tempo ter seu posto reduzido, sendo comparado com a precisão do Matlab e com a redução do posto feita após o uso do CORDIC.

5.2 Filtros de Wiener

Os filtros de Wiener são filtros lineares discretos no tempo, amplamente usados para aplicações de estimação ou predição linear. Sua função custo é o valor médio-quadrático do erro estimado, que é minimizada como critério de otimização estatística [14].

Os filtros de Wiener são também chamados de *estimadores lineares ótimos* no sentido do erro médio quadrático, possuindo a característica de lidar com estatísticas de primeira e segunda ordem do sinal desejado e do ruído aditivo a este sinal [15, 16].

Para um melhor entendimento, vamos considerar um sinal estacionário no sentido amplo $x(n)$. O sinal na saída $y(n)$ de um filtro causal linear invariante no tempo com resposta ao impulso $h(n)$ é dado pela equação:

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k). \tag{5.1}$$

Esse filtro, definido por $h(n)$, gera através de $x(n)$ uma saída que tende ao sinal desejado $s(n)$. O erro ou resíduo $e(n)$ entre o sinal estimado $y(n)$ e o sinal desejado $s(n)$ é dado por:

$$e(n) = s(n) - y(n). \tag{5.2}$$

O filtro de Wiener tem como objetivo manter o erro o menor possível do seu ponto de vista estatístico. Ou seja:

$$\begin{aligned} \epsilon &= E\Big[e^2(n)\Big] \\ &= E\Big[\big(s(n) - y(n)\big)^2\Big] \\ &= E\Big[s^2(n)\Big] - 2E\Big[s(n)y(n)\Big] + E\Big[y^2(n)\Big] \end{aligned} \tag{5.3}$$

Em que:

- O valor médio quadrático do sinal desejado $s(n)$ é igual a $E\Big[s^2(n)\Big]$, ou seja, é a autocorrelação do sinal desejado para *lag* igual a zero ($r_d(0)$).
- O segundo termo $E\Big[\big(s(n) - y(n)\big)^2\Big]$ é a correlação cruzada entre o sinal de entrada do filtro e o sinal desejado, para *lags* iguais a k (que variam de zero ao infinito).
- O terceiro termo é a autocorrelação do sinal de entrada do filtro para *lags* iguais a $(k - m)$, ou seja, $r_x(k - m)$.

Assumindo que o sinal de entrada do filtro $x(n)$ e o sinal desejado $s(n)$ são conjuntamente estacionários no sentido amplo, a correlação cruzada entre eles passa a ser independente do tempo, podendo o erro médio quadrático ser reescrito como:

$$\epsilon = r_d(0) - 2\sum_{k=0}^{\infty} h(k) r_{dx}(k) + \sum_{k=0}^{\infty}\sum_{m=0}^{\infty} h(k)h(m)r_x(k-m). \tag{5.4}$$

Derivando a equação (5.4) em relação a $h(k)$, temos:

$$\frac{\partial \epsilon}{\partial h(k)} = -2r_{dx}(k) + 2\sum_{m=0}^{\infty} h(m) r_x(k-m). \tag{5.5}$$

Igualando a equação (5.5) a zero, é obtida a equação de Wiener-Hopf, em que as únicas quantidades conhecidas são a autocorrelação do sinal de entrada do filtro e a correlação cruzada entre o sinal desejado e o sinal de entrada do filtro.

$$\begin{aligned} -2r_{dx}(k) + 2\sum_{m=0}^{\infty} h_o(m) r_x(k-m) &= 0 \\ \Rightarrow \sum_{m=0}^{\infty} h_o(m) r_x(k-m) &= r_{dx}(k) \end{aligned} \tag{5.6}$$

A variável $h_o(k)$ é o coeficiente do filtro causal que minimiza o erro médio quadrático.

5.3 Variância do erro e Coerência

Antes de abordar cada estágio da decomposição, é necessário entender o conceito de variância do erro e de coerência.

Revendo um antigo problema em predição linear, define-se um vetor aleatório de média zero $\mathbf{x} = \begin{bmatrix} x(1) & x(2) & \dots & x(m) \end{bmatrix}^T = \begin{bmatrix} x(1) & \mathbf{x}^T(1) \end{bmatrix}$. Este vetor possui uma matriz de covariância dada pela fórmula abaixo:

$$\mathbf{R}_{xx} = E\begin{bmatrix} x(1) \\ \mathbf{x}(1) \end{bmatrix}\begin{bmatrix} x(1) & \mathbf{x}^T(1) \end{bmatrix} = \begin{bmatrix} r_{xx}(1) & \mathbf{r}_{xx}^T(1) \\ \mathbf{r}_{xx}(1) & \mathbf{R}_{xx}(1) \end{bmatrix} \tag{5.7}$$

O determinante de $\mathbf{R}_{xx}$ pode ser calculado em função do seu complemento de Schur, que pode ser explicitado pelo escalar:

$$P = r_{xx}(1) - \mathbf{r}_{xx}^T(1)\mathbf{R}_{xx}^{-1}(1)\mathbf{r}_{xx}(1) \tag{5.8}$$

$$\det\begin{bmatrix} r_{xx}(1) & \mathbf{r}_{xx}^T(1) \\ \mathbf{r}_{xx}(1) & \mathbf{R}_{xx}(1) \end{bmatrix} = \det\mathbf{R}_{xx}(1)\det P \tag{5.9}$$

$$= [r_{xx}(1) - \mathbf{r}_{xx}^T(1)\mathbf{R}_{xx}^{-1}(1)\mathbf{r}_{xx}(1)]\det\mathbf{R}_{xx}(1) \tag{5.10}$$

$$= r_{xx}(1)\left[1 - \frac{\mathbf{r}_{xx}^T(1)\mathbf{R}_{xx}^{-1}(1)\mathbf{r}_{xx}(1)}{r_{xx}(1)}\right]\det\mathbf{R}_{xx}(1) \tag{5.11}$$

Podemos escrever a equação (5.11) definindo duas novas variáveis q_{xx} e k_{xx}, sendo:

$$det[\mathbf{R}_{xx}] = q_{xx}(1)det[\mathbf{R}_{xx}(1)] \tag{5.12}$$

em que $q_{xx}(1)$ é a variância do erro ao se estimar o escalar $x(1)$ a partir do vetor $\mathbf{x}(1)$.

$$q_{xx}(1) = r_{xx}(1)[1 - k_{xx}^2(1)] \tag{5.13}$$

$$k_{xx}^2(1) = \frac{\mathbf{r}_{xx}^T(1)\mathbf{R}_{xx}^{-1}(1)\mathbf{r}_{xx}(1)}{r_{xx}(1)} \tag{5.14}$$

A variável $k_{xx}^2(1)$ é definida como a coerência quadrática entre o escalar $x(1)$ e o vetor $\mathbf{x}(1)$ [17], podendo ser escrita como:

$$k_{xx}^2(1) = \mathbf{k}_{xx}(1)\mathbf{k}_{xx}^T(1) \tag{5.15}$$

$$\mathbf{k}_{xx}(1) = r_{xx}^{-1/2}(1)\mathbf{r}_{xx}^T(1)\mathbf{R}_{xx}^{-T/2}(1) \tag{5.16}$$

O vetor $\mathbf{k}_{xx}(1)$ é a coerência entre $x(1)$ e $\mathbf{x}(1)$ ou a correlação cruzada entre a variável aleatória branca $r_{xx}^{-1/2}(1)x(1)$ e o vetor aleatório branco $\mathbf{R}_{xx}^{-1/2}\mathbf{x}(1)$:

$$\mathbf{k}_{xx}(1) = E\left[r_{xx}^{-1/2}(1)x(1)\mathbf{x}^T(1)\mathbf{R}_{xx}^{-T/2}(1)\right] \tag{5.17}$$

$$= r_{xx}^{-1/2}(1)\mathbf{r}_{xx}^T(1)\mathbf{R}_{xx(1)}^{-T/2} \tag{5.18}$$

Uma vez definidas as variáveis de coerência e da variância do erro, pode-se escrever o determinante $det[\mathbf{R}_{xx}]$ como:

$$det[\mathbf{R}_{xx}] = \prod_{i=1}^{m} q_{xx}(i) \tag{5.19}$$

$$= \prod_{i=1}^{m} r_{xx}(i)[1 - k_{xx}^2(i)] \tag{5.20}$$

$$k_{xx}^2(i) = \frac{\mathbf{r}_{xx}^T(i)\mathbf{R}_{xx}^{-1}(i)\mathbf{r}_{xx}(i)}{r_{xx}(i)} \tag{5.21}$$

Sendo $k_{xx}^2(i)$ a coerência quadrática entre o escalar $x(i)$ e o vetor $\mathbf{x}(i) = \begin{bmatrix} x(i+1) & \dots & x(m) \end{bmatrix}^T$. A equação (5.19) é o determinante de Gram, com cada variância do erro de predição descrita em termos da coerência quadrática.

Seguindo na mesma linha de raciocínio, o filtro de Wiener também pode ser descrito em função da coerência, dada por:

$$\mathbf{w}(i) = \mathbf{r}_{xx}^T(i)\mathbf{R}_{xx}^{-1}(i) = r_{xx}^{1/2}(i)\mathbf{k}_{xx}(i)\mathbf{R}_{xx}^{-1/2}(i) \tag{5.22}$$

5.4 Coordenadas padrão

No contexto de filtragem, a figura abaixo mostra um vetor fonte $\mathbf{x}_{m\times 1}$ e um vetor $\mathbf{y}_{n\times 1}$ gerados pela *Mãe Natureza*. O *Pai Natureza* só enxerga o vetor medido $\mathbf{y}$, e tem esperanças de estimar a partir das informações que recebeu o sinal $\mathbf{x}$ enviado pela *Mãe Natureza*.

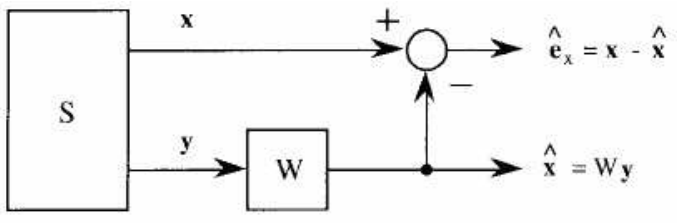

Figura 5.1: Exemplo de Estimação Linear

Podemos escrever a matriz de covariância $\mathbf{R}_{zz}$ como:

$$\mathbf{R}_{zz} = E\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}\begin{bmatrix}\mathbf{x}^T & \mathbf{y}^T\end{bmatrix} = \begin{bmatrix}\mathbf{R}_{xx} & \mathbf{R}_{xy}\\ \mathbf{R}_{xy}^T & \mathbf{R}_{yy}\end{bmatrix} \tag{5.23}$$

O modelo representado na Figura 5.1 mostra o estimador linear MMSE (***M**inimum **M**ean **S**quare **E**rror*) de $\mathbf{x}$ a partir de $\mathbf{y}$, dado pela função de transferência $\hat{\mathbf{x}} = \mathbf{W}\mathbf{y}$. O erro estimado entre os dois sinais é $\hat{\mathbf{e}}_x = \mathbf{x} - \hat{\mathbf{x}}$, e define-se o filtro de Wiener $\mathbf{W}$ e a matriz de covariância do erro $\mathbf{Q}_{xx}$ no sistema de coordenadas padrão de acordo com as equações abaixo:

$$\mathbf{W} = \mathbf{R}_{xy}\mathbf{R}_{yy}^{-1} \tag{5.24}$$

$$\mathbf{Q}_{xx} = E[(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^T] = \mathbf{R}_{xx} - \mathbf{R}_{xy}\mathbf{R}_{yy}^{-1}\mathbf{R}_{xy}^T \tag{5.25}$$

A matriz de covariância do erro, dada pela autocorrelação entre $\mathbf{x}$ e $\hat{\mathbf{x}}$, pode também ser deduzida a partir do complemento de Schur da matriz de covariância $\mathbf{R}_{zz}$.

A figura abaixo ilustra o filtro de Wiener no sistema de coordenadas padrão:

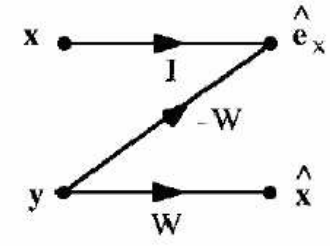

Figura 5.2: Filtro de Wiener no sistema de coordenadas padrão

A transformação linear conduz os vetores fonte e de medição $\mathbf{x}$ e $\mathbf{y}$ aos vetores ortogonais $\hat{\mathbf{e}}_x$ e $\mathbf{y}$, com suas respectivas covariâncias $\mathbf{Q}_{xx}$ e $\mathbf{R}_{yy}$ dadas por:

$$\begin{bmatrix} \hat{\mathbf{e}}_x \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{-W} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \tag{5.26}$$

$$\begin{bmatrix} \mathbf{Q}_{xx} & 0 \\ 0 & \mathbf{R}_{yy} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{-W} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R}_{xx} & \mathbf{R}_{xy} \\ \mathbf{R}_{xy}^T & \mathbf{R}_{yy} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{-W}^T & \mathbf{I} \end{bmatrix} \tag{5.27}$$

Observando a equação (5.26) e aplicando a decomposição de Schur vista na Seção 5.3, pode-se escrever o $det[\mathbf{R}_{zz}]$ como:

$$det[\mathbf{R}_{zz}] = det[\mathbf{Q}_{xx}]det[\mathbf{R}_{yy}] \tag{5.28}$$

$$det[\mathbf{Q}_{xx}] = det[\mathbf{R}_{xx} - \mathbf{R}_{xy}\mathbf{R}_{yy}^{-1}\mathbf{R}_{xy}^T] \tag{5.29}$$

5.5 Redução de Posto

Para determinar os coeficientes do filtro de Wiener, é necessário efetuar o cálculo de matrizes inversas e multiplicar matrizes entre si, ou seja, é preciso compilar algoritmos de complexidade assintótica superior a $O(n^3)$.

Uma maneira de diminuir esta complexidade seria reduzir o posto do estimador ótimo. Para isso, a SVD identifica quais são os termos menos correlatados, para que sejam purgados os menores valores singulares e reduzido o posto das matrizes em operação. Neste caso também é feita a quantização, alocando bits para uma filtragem de Wiener com precisão finita. O objetivo é minimizar o traço da matriz de covariância do erro $\mathbf{Q}_{xx}$, pois o erro médio quadrático estará sendo minimizado.

A redução do posto pode ser feita de duas maneiras: durante e após o uso do CORDIC. Os dois métodos foram descritos no Capítulo 4 (ver Seção 4.3), e serão comparados caso a caso.

Pode-se deduzir a matriz de covariância do erro descrita após a aplicação da SVD:

$$\mathbf{W} = \mathbf{R}_{xy}\mathbf{V}\Sigma^{-1}\mathbf{U}^T \tag{5.30}$$

$$\mathbf{Q}_{xx} = \mathbf{R}_{xx} - \mathbf{R}_{xy}\mathbf{V}\Sigma^{-1}\mathbf{U}^T\mathbf{R}_{xy} \tag{5.31}$$

5.6 Implementação: Filtro de Wiener e redução de posto com precisão variável no Matlab

O objetivo principal desta implementação é comparar a complexidade do algoritmo de redução de posto com precisão do Matlab e com precisão variável dada pelo CORDIC, exercitando os conceitos vistos ao longo deste capítulo. O programa de testes a seguir foi elaborado no Matlab, visando esta comparação:

- **Filtro de Wiener com precisão do Matlab, posto completo** $\longrightarrow$ Implementar o filtro de Wiener aplicando a função SVD do Matlab na matriz de autocorrelação $\mathbf{R}_{yy}$; o arquivo desenvolvido no Matlab chama-se *prec_inf_full.m*.

- **Filtro de Wiener com precisão do Matlab, posto reduzido** $\longrightarrow$ Implementar o filtro de Wiener aplicando a função SVD do Matlab na matriz de autocorrelação $\mathbf{R}_{yy}$, reduzindo o seu posto; o arquivo *prec_inf_red.m* descreve esta implementação.

- **Filtro de Wiener com precisão variável CORDIC, posto completo** $\longrightarrow$ Implementar o filtro de Wiener aplicando a função SVD_CORDIC na matriz de autocorrelação $\mathbf{R}_{yy}$, variando o número de bits de precisão; o arquivo referente no Matlab é o *prec_fin_full.m*.

- **Filtro de Wiener com precisão variável CORDIC, posto reduzido "*a posteriori*"** $\longrightarrow$ Implementar o filtro de Wiener aplicando a função SVD_CORDIC na matriz de autocorrelação $\mathbf{R}_{yy}$, variando o número de bits de precisão e reduzindo o seu posto posteriormente; para isto foi desenvolvido no Matlab o arquivo *prec_fin_red.m*.

- **Filtro de Wiener com precisão variável CORDIC, posto reduzido em conjunto** $\longrightarrow$ Implementar o filtro de Wiener aplicando a função

SVD_CORDICRED na matriz de autocorrelação $\mathbf{R}_{yy}$, variando o número de bits de precisão e reduzindo o seu posto "ao mesmo tempo"; para isto foi desenvolvido no Matlab o arquivo *prec_fin_rediterativo.m*.

Para todas as implementações geramos um sinal $\mathbf{x}_{n\times 1}$ com distribuição $\mathcal{N}(0,1)$. Este sinal é filtrado por um canal ruidoso (ruído branco e gaussiano) gerando o sinal $\mathbf{y}_{n\times 1}$. O canal foi feito pela função *filter* do Matlab, com coeficientes iguais a [1 1 0.5].

Com os dois sinais $\mathbf{x}$ e $\mathbf{y}$, estimam-se correlações $\mathbf{R}_{xx}, \mathbf{R}_{yy}$ e $\mathbf{R}_{xy}$ através de sua média estatística.

5.6.1 Wiener com precisão do Matlab e posto completo

Uma vez definidas as matrizes $\mathbf{R}_{yy}$ e $\mathbf{R}_{xy}$, o filtro de Wiener $\mathbf{W}_{n\times n}$ é dado pela fórmula (5.24). Na implementação, o filtro é decomposto em três estágios dados pelas matrizes da decomposição em valores singulares, com precisão do Matlab. Esta decomposição será usada *a posteriori* na redução do posto da matriz $\mathbf{R}_{yy}$, conforme realizada no trabalho de Scharf [17] (porém com a SVD aplicada na matriz $\mathbf{R}_{yy}$). A matriz é reconstruída pela multiplicação dos três estágios $\mathbf{S}$, $\mathbf{V}$ e $\mathbf{D}^T$, e posteriormente é calculado o erro médio quadrático representado pelo traço da matriz $\mathbf{Q}_{xx}$.

Na figura abaixo pode-se ver a norma do erro acumulado pela quantização do Matlab dado por $\|\hat{\mathbf{e}}\| = \|\mathbf{x} - \hat{\mathbf{x}}\|$ para 64 amostras de estimação nas coordenadas padrão:

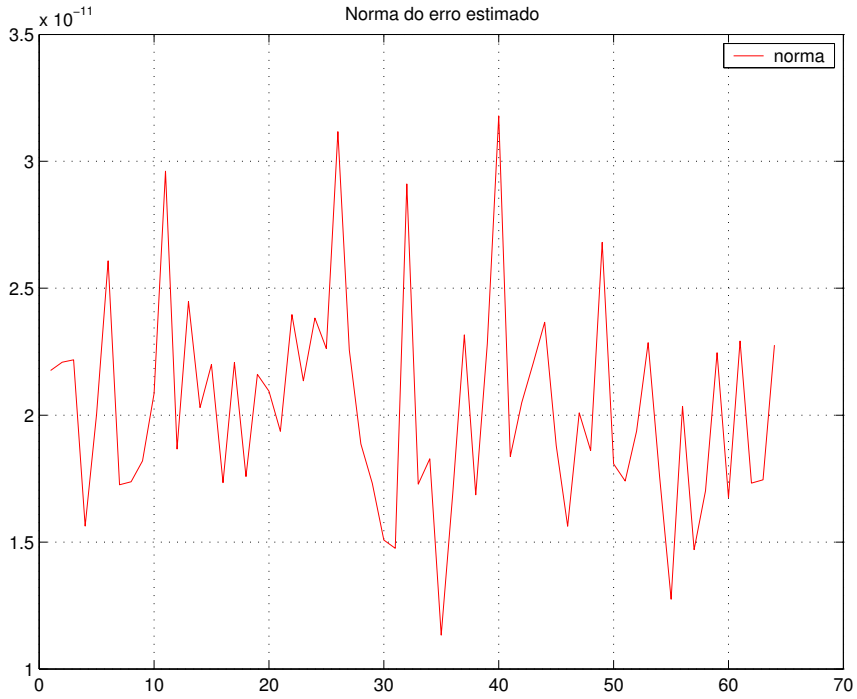

Figura 5.3: Norma do erro acumulado pela quantização do Matlab

A norma do erro e o traço da matriz de covariância do erro $\mathbf{Q}_{xx}$ são mostrados na tabela abaixo:

Tabela 5.1: Norma do erro e erro médio quadrático com precisão do Matlab

Coordenadas padrão	
$\|\hat{\mathbf{e}}\| = \|\mathbf{x} - \hat{\mathbf{x}}\|$	$7,2148 \cdot 10^{-11}$
$\mathrm{tr}[\mathbf{Q}_{xx}]$	$2,1817 \cdot 10^{-12}$

Devido à ordem de grandeza da norma do erro, o gráfico dos dois vetores $\mathbf{x}$ e $\hat{\mathbf{x}}$ não apresenta diferenças entre si na escala do gráfico. Fato que mudará quando o posto for reduzido e a precisão dos cálculos do filtro de Wiener for escolhida.

A complexidade assintótica do algoritmo no que tange a geração do filtro de Wiener é da ordem de $O(n^3)$, pois as matrizes possuem o posto completo.

5.6.2 Wiener com precisão do Matlab e posto reduzido

Como visto no capítulo 3, a decomposição em valores singulares permite analisar visualmente a influência de cada valor singular no posto da matriz desejada. No caso da simulação anterior, a matriz $\mathbf{V}$ formada pelos valores singulares de $\mathbf{R}_{yy}$ pode ser expressa através do gráfico abaixo:

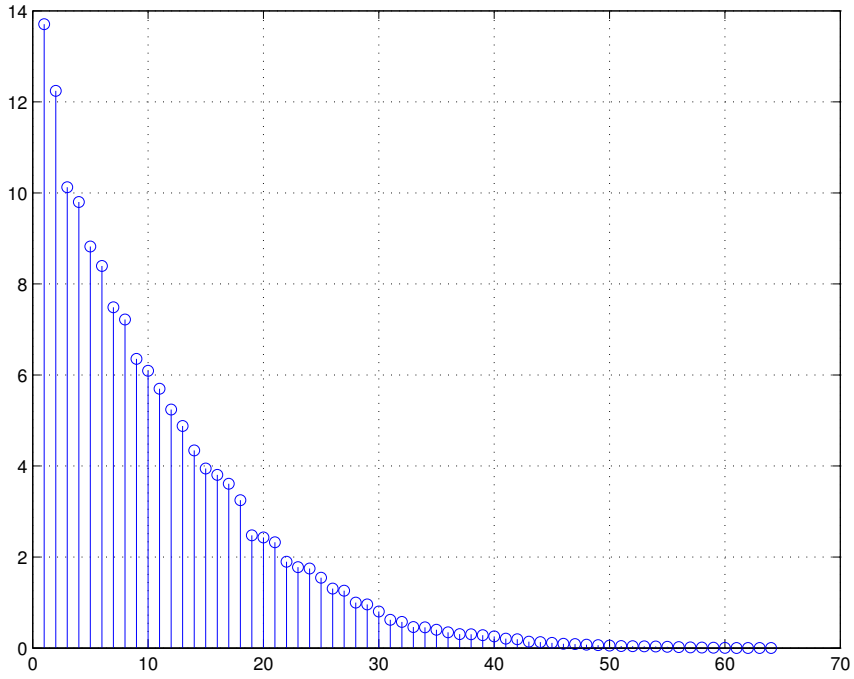

Figura 5.4: Valores singulares de $\mathbf{R}_{yy}$

Dado que se quer minimizar a complexidade do algoritmo através da redução de posto do filtro de Wiener (neste caso com os erros de precisão dados pelo Matlab), espera-se que, se os valores singulares muito menores que o maior valor singular fossem expurgados, haveria uma aproximação aceitável da matriz original e a redução considerável da complexidade do algoritmo. A seguir um exemplo da redução do posto na matriz $\mathbf{R}_{yy}$, para diferentes valores do posto:

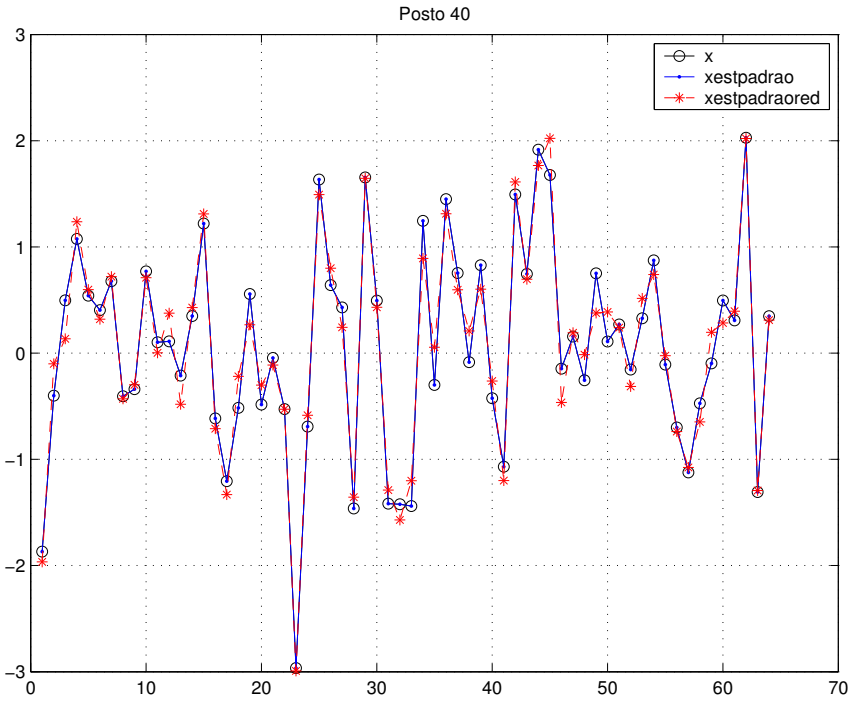

Figura 5.5: Figura referente a Posto 40

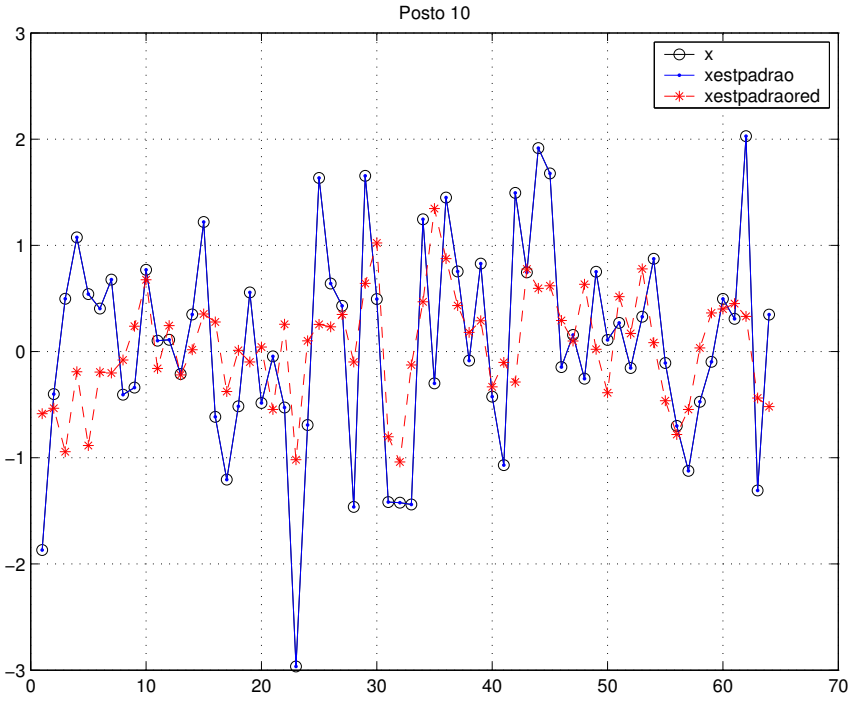

Figura 5.6: Figura referente a Posto 10

Nas figuras acima, geramos o vetor original $\mathbf{x}$, o vetor estimado $\hat{\mathbf{x}}$ e o vetor estimado com posto reduzido $\hat{\mathbf{x}}_{red}$. Vemos que entre $\mathbf{x}$ e $\hat{\mathbf{x}}$ o erro é imperceptível

dentro da escala dos gráficos, sendo os dois sinais quase idênticos.

O resultado com redução de posto para 40 quase não altera o erro entre os dois sinais, tendo um MSE pequeno (tabela abaixo). Não se pode dizer o mesmo da Figura 5.6, que para um posto 10 apresenta um MSE bem maior. O *posto ideal* para cada aplicação dependerá da taxa de distorção admitida pelo sistema.

A figura a seguir mostra a evolução do erro médio quadrático ao longo da redução de posto.

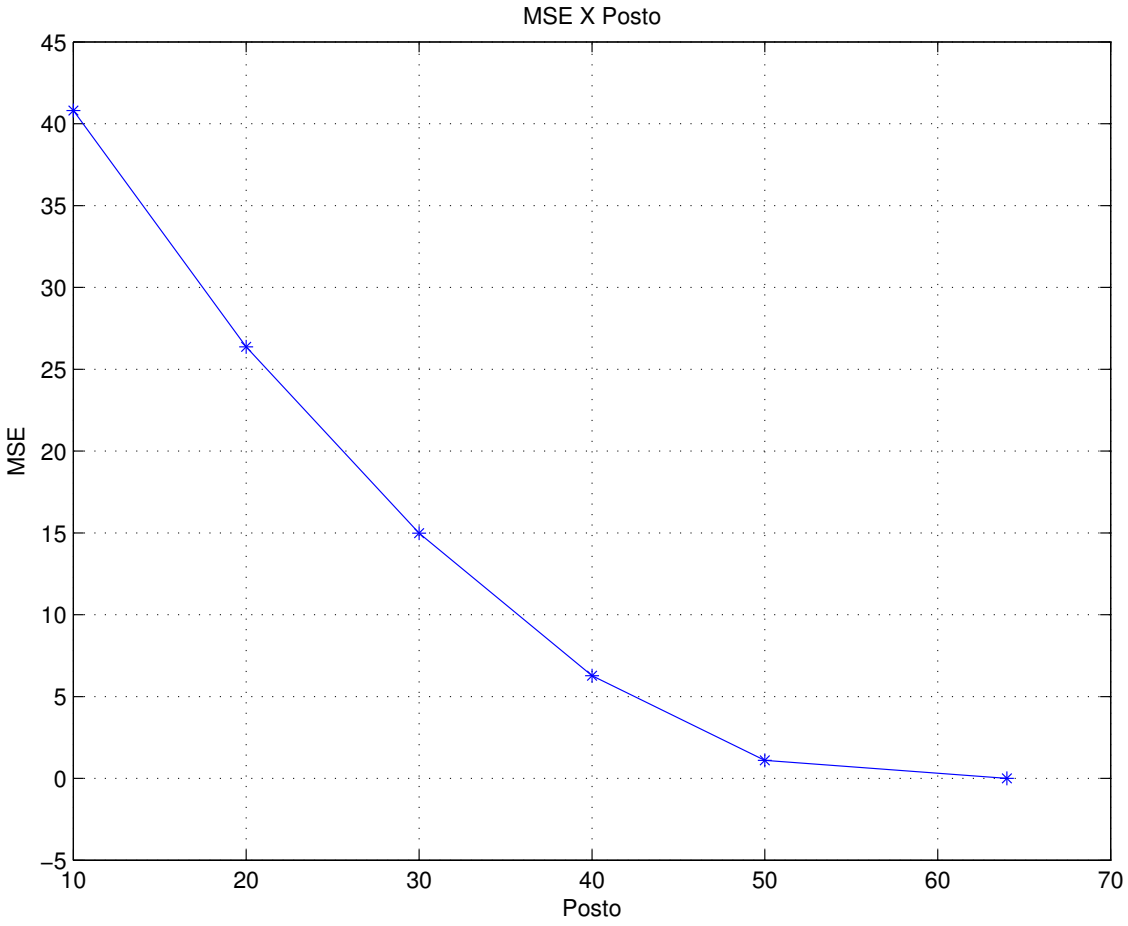

Figura 5.7: MSE × Posto de $\mathbf{R}_{yy}$

Quando se reduz o posto de uma matriz, a complexidade assintótica do algoritmo cai de $O(n^3)$ para a ordem de $O(nr^2)$, em que $\mathbf{r}$ é o novo posto reduzido. Independente do hardware ou do compilador utilizados, reduzir o posto de matrizes de ordens elevadas é um grande artifício para reduzir o tempo de execução na aplicação.

Tabela 5.2: Erro médio quadrático com precisão do Matlab para diferentes postos reduzidos

Coordenadas padrão	
Posto	$\mathrm{tr}[\mathbf{Q}_{xx}]$
64	$2,1817 \cdot 10^{-12}$
50	$1,1063$
40	6,2726
30	$14,9783$
20	26,3720
10	40,8101

5.6.3 Wiener com precisão variável pela *SVD_CORDIC* e posto completo

Quando se calcula o filtro de Wiener com precisão do Matlab, só existem erros de filtragem intrínsecos à quantização no algoritmo. A ferramenta SVD_CORDIC permite a decomposição da SVD com precisão de n bits, gerando erros de quantização.

A seqüência de figuras a seguir mostra a estimação de $\mathbf{x}$ com 10, 7, e 5 bits de precisão no cálculo da SVD.

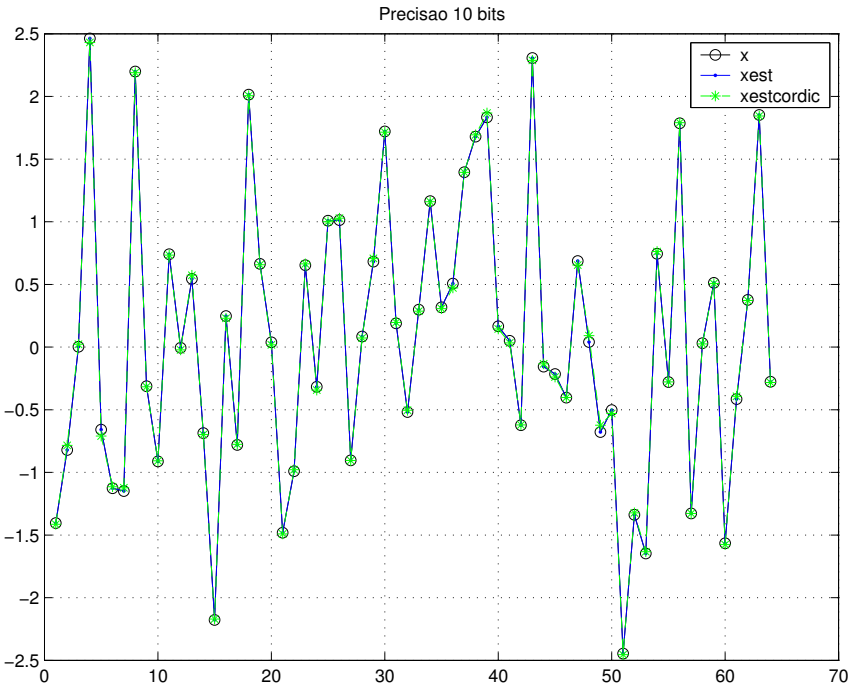

Figura 5.8: Wiener com 10 bits de precisão

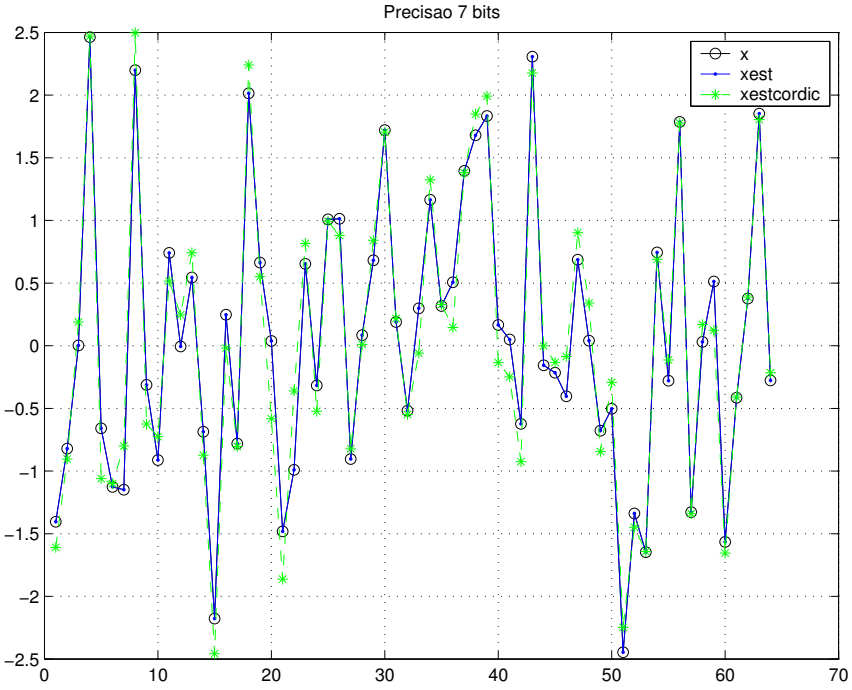

Figura 5.9: Wiener com 7 bits de precisão

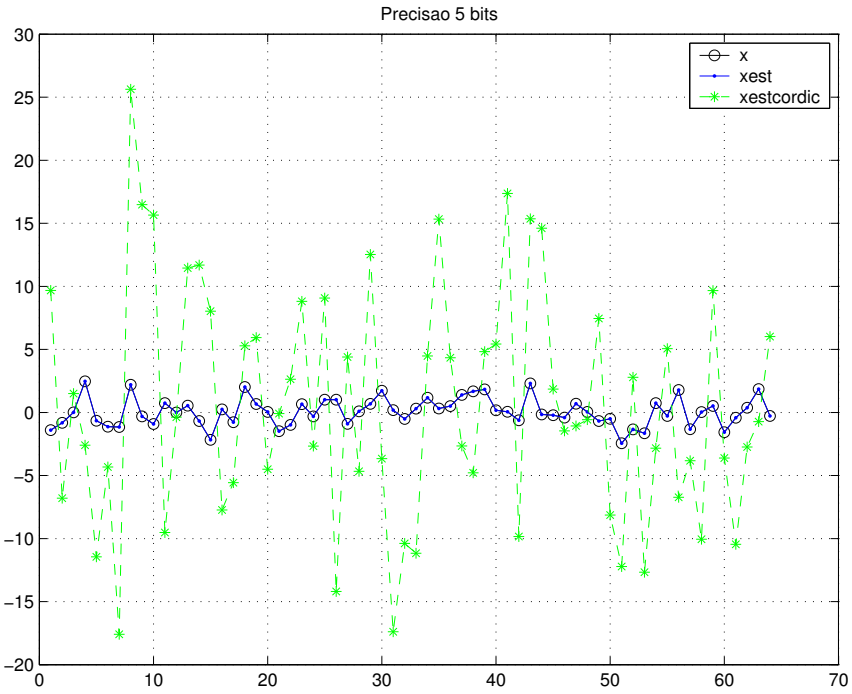

Figura 5.10: Wiener com 5 bits de precisão

Pode-se observar na Figura 5.8 que o valor de $\hat{\mathbf{x}}$ para 10 bits de precisão é muito próximo ao valor calculado com a precisão do Matlab, sendo imperceptível na escala adotada.

Na figura referente a 7 bits de precisão (Figura **??**), a distância entre a estimação do Matlab e a dada pelo CORDIC começa a aumentar. À medida que os bits de precisão vão diminuindo, o MSE vai aumentando gradativamente. É importante ressaltar que está havendo um erro causado pela precisão CORDIC que se propaga na equação (5.24), impossibilitando trabalhar com precisão inferior a 5 bits no caso desta implementação. Os resultados obtidos no Capítulo 4 mostram o erro isoladamente (sem se propagar no algoritmo) obtido pela função SVD_CORDIC para 3, 5, 7, 10 e 15 bits de precisão. Mesmo assim, é mais vantajoso do ponto de vista computacional usar a menor quantidade de bits possíveis para atingirmos uma acurácia aceitável.

A tabela abaixo mostra a evolução do erro médio quadrático ao longo dos bits de precisão alocados para a SVD do exemplo acima.

O erro médio quadrático vai crescendo de acordo com a alocação de bits para o cálculo da SVD, conforme esperado. A distorção máxima aceita pelo sistema

Tabela 5.3: Erro médio quadrático com precisão variável dada pelo CORDIC

Coordenadas padrão	
Bits de Precisao	$\mathrm{tr}[\mathbf{Q}_{xx}]$
Matlab	$1,6320 \cdot 10^{-14}$
10	0,0641
8	3,3760
7	6,0276
5	103,2396
3	248436,4685

dependerá dos número de bits de precisão.

5.6.4 Wiener com precisão variável pela *SVD_CORDIC* e posto reduzido "*a posteriori*"

Quando se calcula o filtro de Wiener com precisão CORDIC e se reduz o seu posto, são introduzidas duas variáveis de erro: O erro "*bias-squared*" introduzido pela redução do posto e a variância introduzida pela quantização.

Para entender o resultado da simulação, o número de bits de precisão será fixado e a redução de posto será variada gradativamente, para que seja vista a comparação entre os vetores $\hat{\mathbf{x}}_{red}$ (que representa a redução de posto com precisão do Matlab) e o vetor $\hat{\mathbf{x}}_{cordicred}$ (que representa a redução de posto com precisão CORDIC de n bits). Esta seqüência será repetida para 10, 8 e 5 bits de precisão, reduzindo o posto de 64 para 50, 40, 30, 20 e 10.

5.6.4.1 10 bits de precisão

Conforme visto nas Seções 5.6.1 e 5.6.3, as estimações com precisão do Matlab e com CORDIC de 10 bits para posto 64 são quase idênticas ao vetor original.

A seqüência de figuras a seguir mostra o vetor estimado $\hat{\mathbf{x}}$ com 10 bits de precisão, para diferentes reduções de posto na matriz $\mathbf{R}_{yy}$:

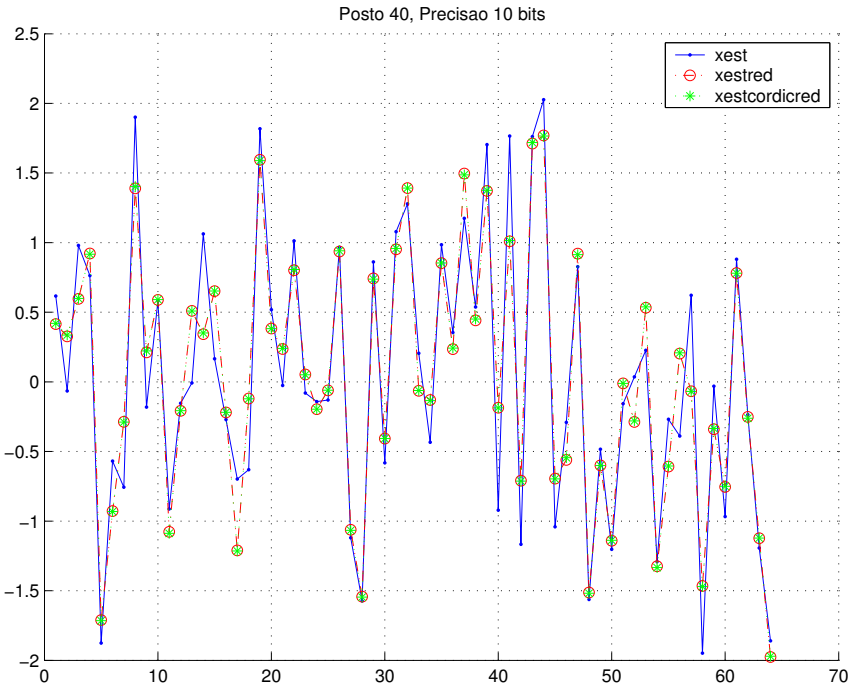

Figura 5.11: Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 10 bits

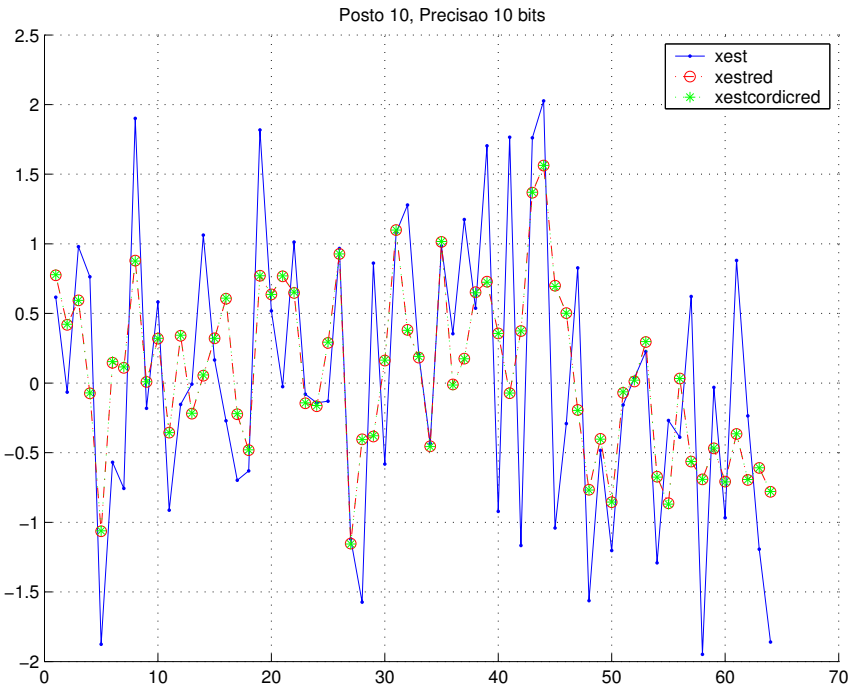

Figura 5.12: Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 10 bits

Em todas as figuras anteriores, independente do posto reduzido, o vetor referente à precisão CORDIC de 10 bits é muito próximo ao vetor com precisão do Matlab, conforme esperado. Isto significa que para 10 bits de precisão o resultado é bastante satisfatório e apresenta um erro médio quadrático baixo.

A tabela abaixo contém os erros médios quadráticos para todos os postos com 10 bits de precisão CORDIC e com a precisão do Matlab:

Tabela 5.4: Erro médio quadrático com precisão CORDIC de 10 bits e precisão do Matlab para diferentes postos reduzidos

tr[$\mathbf{Q}_{xx}$] - Redução com Precisão 10 bits		
Posto	Após o CORDIC	Precisão Matlab
64	0,6238	0,0000
50	1,1444	1,1039
40	5,4586	5,4263
30	13,7176	13,6775
20	24,7496	24,7061
10	39,1946	39,1672

Nota-se que quando se reduz o posto gradativamente, o erro médio quadrático também vai aumentando, conforme o esperado. Dependendo da taxa de distorção mínima desejável para o sistema, é possível eliminar mais de 20 valores singulares da matriz, com precisão dada pelo CORDIC e com resultados satisfatórios.

5.6.4.2 8 bits de precisão

Como a precisão agora é de 8 bits, aumentou a distância entre o vetor estimado pelo CORDIC $\hat{\mathbf{x}}_{estcordic}$ e o vetor fonte $\mathbf{x}$, como vimos na seção 5.6.3. Quando se reduz o posto gradativamente, obtem-se:

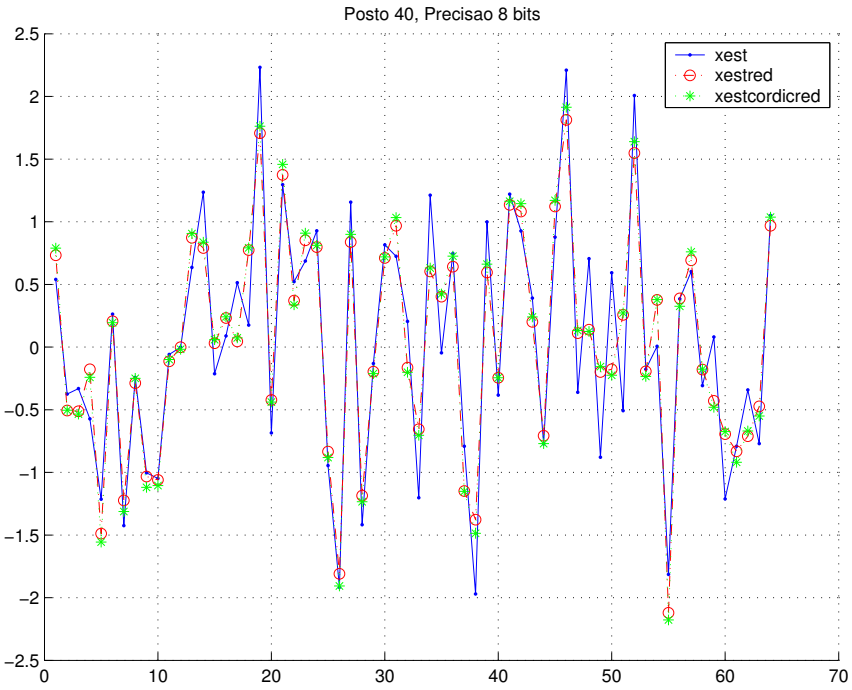

Figura 5.13: Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 8 bits

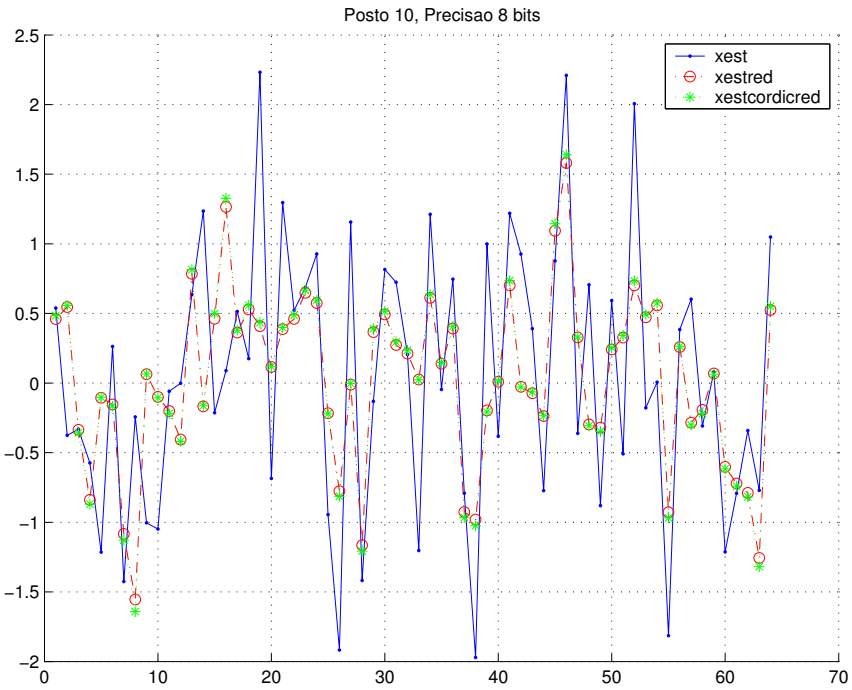

Figura 5.14: Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 8 bits

Nas figuras anteriores com precisão CORDIC de 8 bits, observa-se que o vetor $\hat{\mathbf{x}}_{estcordicred}$ se aproxima do seu vetor gabarito $\hat{\mathbf{x}}_{estred}$ (precisão Matlab, posto reduzido), e a distância entre eles é devida somente ao erro de quantização com 8 bits. Pode-se notar que tanto o vetor $\hat{\mathbf{x}}_{estcordicred}$ quanto o vetor $\hat{\mathbf{x}}_{estred}$ vão se afastando do vetor original $\mathbf{x}$ à medida que o posto vai sendo reduzido.

A tabela a seguir contém os erros médios quadráticos para todos os postos com 8 bits de precisão CORDIC e precisão Matlab:

Tabela 5.5: Erro médio quadrático com precisão CORDIC de 8 bits e precisão do Matlab para diferentes postos reduzidos

tr[$\mathbf{Q}_{xx}$] - Redução com Precisão 8 bits		
Posto	Após o CORDIC	Precisão Matlab
64	3,8470	0,0000
50	2,1155	1,1137
40	3,0104	5,9709
30	12,1202	14,5821
20	24,0106	25,7785
10	39,8054	40,9278

O fato das microrotações CORDIC apresentarem imperfeições devido a operações trigonométricas explica a pequena diferença entre os valores do Matlab e CORDIC.

Um outro ponto interessante é o fato do posto 50 e 40 terem apresentado valores com erro menor do que o valor com posto completo. Isto é explicado devido à propagação do erro CORDIC no algoritmo de estimação de Wiener nas coordenadas padrão. Quando se faz a redução de posto, apesar do erro "*bias-squared*" ser introduzido, retira-se, além da informação dos menores valores singulares, algumas incertezas originadas nas microrotações, fazendo com que o valor do CORDIC para pequenas reduções de posto seja ligeiramente melhor do que o valor apresentado para posto completo.

À medida que os erros de quantização vão aumentando devido ao menor número de bits de precisão, este efeito se torna mais visível. Um exemplo é a

estimação para 5 bits de precisão CORDIC, como será visto na próxima seção.

5.6.4.3 5 bits de precisão

Como a precisão agora é de 5 bits, aumentou ainda mais a distância entre o vetor estimado pelo CORDIC $\hat{\mathbf{x}}_{estcordic}$ e o vetor fonte $\mathbf{x}$.

Para exemplificar o efeito com o posto reduzido, geramos as figuras abaixo:

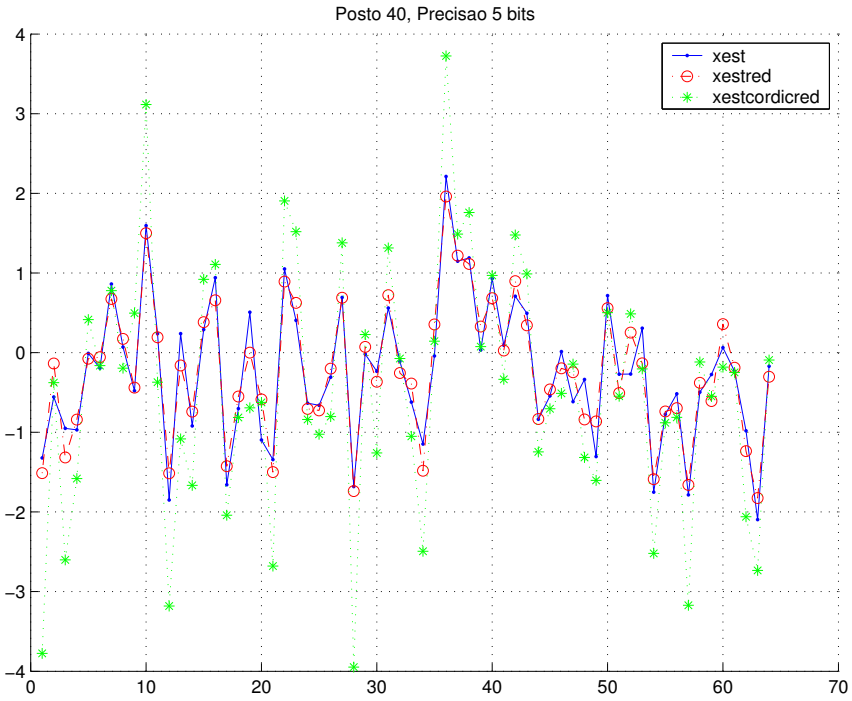

Figura 5.15: Wiener para posto 40, comparando as precisões do Matlab e CORDIC de 5 bits

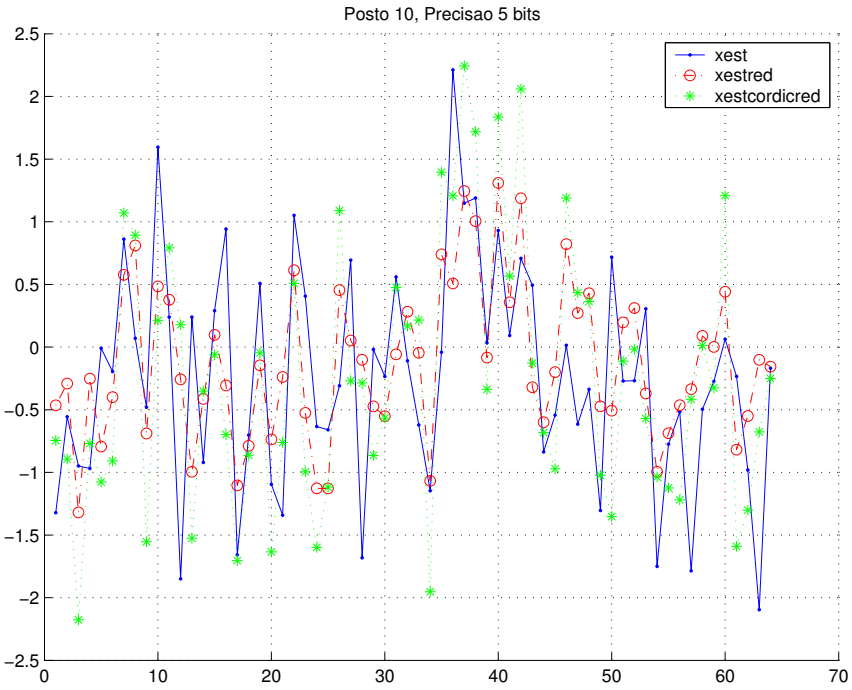

Figura 5.16: Wiener para posto 10, comparando as precisões do Matlab e CORDIC de 5 bits

Nas figuras anteriores com precisão CORDIC de 5 bits, observa-se que $\hat{\mathbf{x}}_{estcordicred}$ (precisão CORDIC, posto reduzido) não consegue acompanhar o seu vetor gabarito $\hat{\mathbf{x}}_{estred}$ (precisão Matlab, posto reduzido) devido ao erro de quantização com 5 bits. Pode-se observar que tanto o vetor $\hat{\mathbf{x}}_{estcordicred}$ quanto o vetor $\hat{\mathbf{x}}_{estred}$ vão se afastando do vetor original $\mathbf{x}$ à medida que o posto vai sendo reduzido.

A precisão numérica da estimação CORDIC está impedindo que uma análise mais profunda possa ser feita, prejudicando o comportamento do sistema. Como os erros são propagados, a relação entre o posto e o erro médio quadrático é diretamente afetada, conforme visto na tabela a seguir:

Tabela 5.6: Erro médio quadrático com precisão CORDIC de 5 bits e precisão do Matlab para diferentes postos reduzidos

tr[$\mathbf{Q}_{xx}$] - Redução com Precisão 5 bits		
Posto	Após o CORDIC	Precisão Matlab
64	228,5394	0,0000
50	53,9633	0,9079
40	43,6135	5,4740
30	26,8654	13,5014
20	5,3802	24,3718
10	24,0215	39,2494

Há de se ressaltar que o erro médio quadrático da estimação CORDIC é bastante elevado para o posto completo, inviabilizando qualquer tentativa de estimação. Isto se deve à propagação do erro da ordem de 5 bits na matriz de Givens, que é refletida no algoritmo de Wiener. O erro é então multiplicado por toda a decomposição de Schur e repassado integralmente na equação (5.24), o que deixa no caso desta aplicação o resultado para valores com menos de 5 bits de precisão CORDIC imprecisos e não satisfatórios.

5.6.5 Wiener com precisão variável pela *SVD_CORDICRED* e posto reduzido em conjunto

Nesta simulação, o número de bits de precisão será fixado e a redução de posto será variada gradativamente. A diferença ocorre na redução do posto realizada logo após o primeiro bit de precisão. Depois da redução do posto, a matriz continua a ser rotacionada com o restante dos bits de precisão desejados.

Os gráficos mostrarão a comparação entre os vetores $\hat{\mathbf{x}}_{estred}$ (que representa a redução de posto com precisão do Matlab), $\hat{\mathbf{x}}_{cordicred}$ (que representa a redução de posto após o primeiro bit de precisão CORDIC) e o vetor $\hat{\mathbf{x}}_{cordicred2}$ (que representa a redução de posto depois de aplicada a decomposição com os n bits de precisão CORDIC). Esta seqüência será repetida para 10 e 8 bits de precisão, e para o posto

de 50, 45, 40, 35 e 30.

5.6.5.1 10 bits de precisão

As figuras a seguir comparam os vetores estimados $\hat{\mathbf{x}}_{estred}$, $\hat{\mathbf{x}}_{cordicred}$ (durante CORDIC) e $\hat{\mathbf{x}}_{cordicred2}$ (após CORDIC) com 10 bits de precisão, para diferentes reduções de posto na matriz R_{yy}:

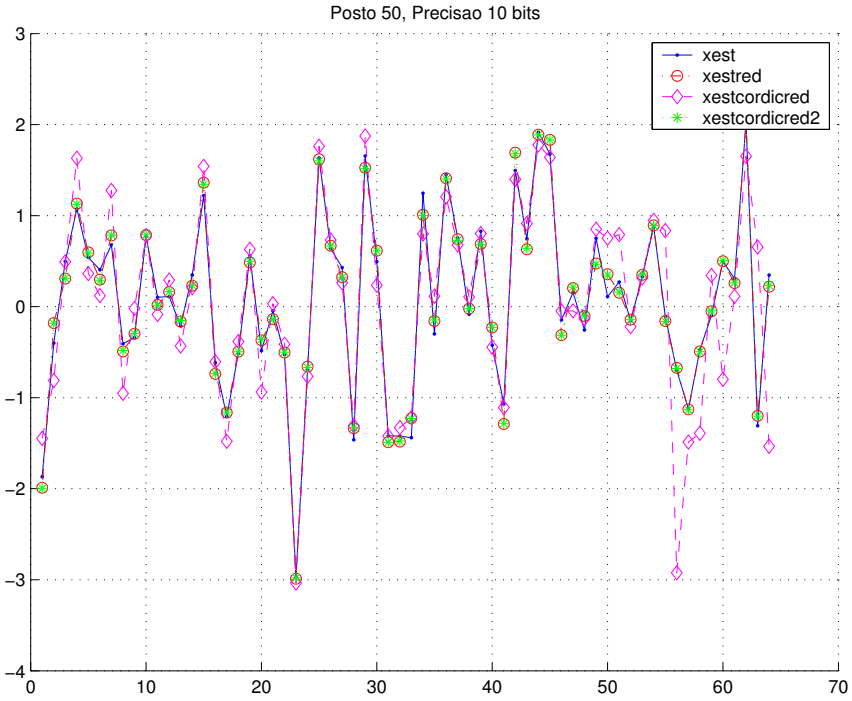

Figura 5.17: Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 10 bits e posto 50

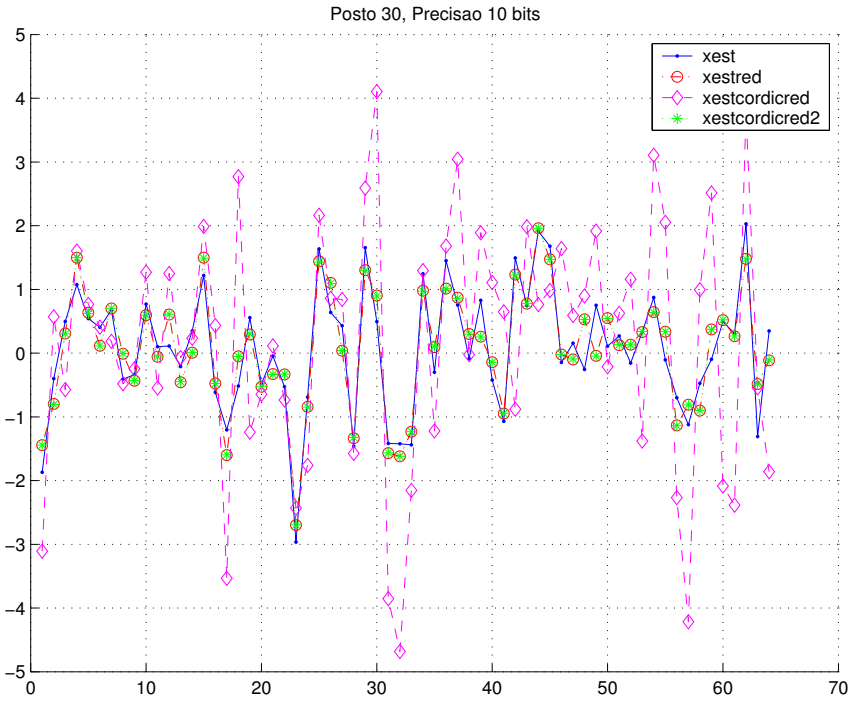

Figura 5.18: Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 10 bits e posto 30

A tabela abaixo contém os erros médios quadráticos para cada vetor comparado:

Tabela 5.7: Erro médio quadrático com precisão CORDIC de 10 bits e precisão do Matlab para diferentes postos reduzidos

tr[$\mathbf{Q}_{xx}$] - Redução com 10 Bits de Precisão			
Posto	Durante o CORDIC	Após o CORDIC	Precisão Matlab
64	0,2240	0,2422	0,0000
50	1,1606	1,0415	0,9824
45	14,6476	2,8045	2,7496
40	7,3465	5,7189	5,6646
35	15,0949	9,5335	9,4774
30	22,2999	13,4817	13,4479

Nota-se que quando o posto é reduzido gradativamente, o erro médio quadrático também vai aumentando para todos os métodos, conforme o esperado.

Em todas as figuras acima para 10 bits, o vetor referente ao cálculo simultâneo da precisão e redução de posto (método 1, vetor $\hat{\mathbf{x}}_{cordicred}$) apresenta um MSE um pouco maior que o método 2, representado pelo vetor $\hat{\mathbf{x}}_{cordicred2}$). Isto é explicado porque existe um erro de quantização intrínseco ao método 1 que é introduzido quando ocorre a primeira rotação de Jacobi, com 1 bit de precisão. Quando ocorre a redução de posto, o erro "*bias-squared*" é então introduzido na simulação, mas em valor absoluto maior do que o erro de quantização, o que faz com que o MSE vá aumentando com a redução do posto independente do método.

É importante comentar que conforme foi visto na Seção 5.6.4, o método 2 para 10 bits de precisão apresenta resultados parecidos com os resultados da precisão do Matlab, ou seja, para 10 bits de precisão o resultado é satisfatório e apresenta um erro médio quadrático compatível à precisão do Matlab.

5.6.5.2 8 bits de precisão

Para 8 bits de precisão, geramos as seguintes figuras para posto reduzido:

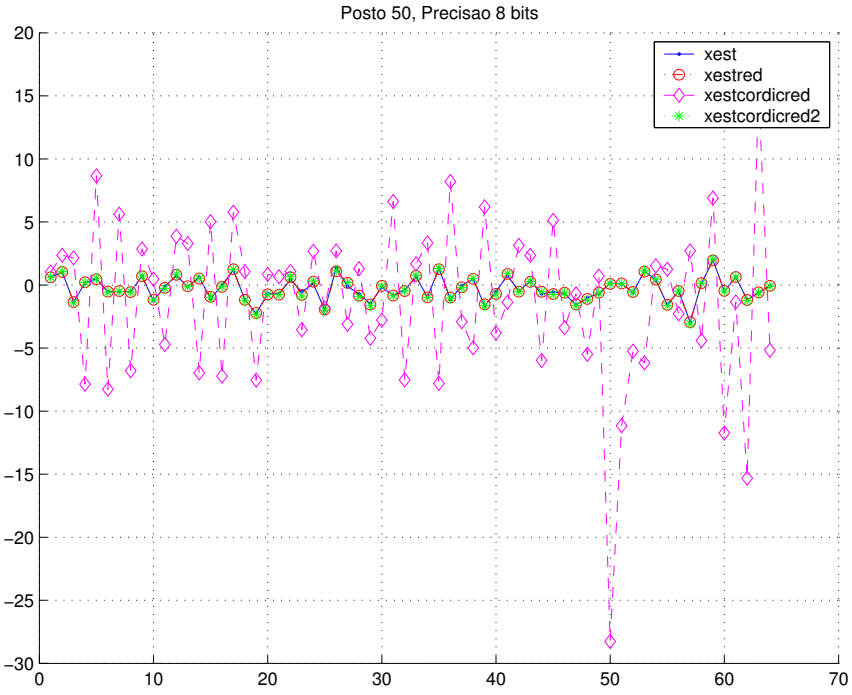

Figura 5.19: Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 8 bits e posto 50

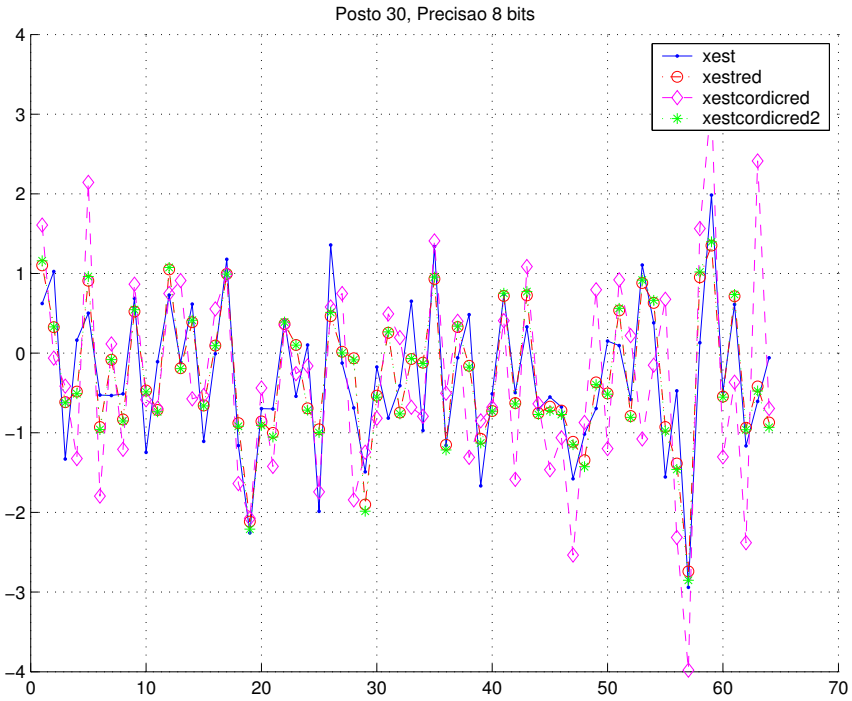

Figura 5.20: Wiener com posto reduzido enquanto a precisão CORDIC é calculada (método 1) e depois do CORDIC (método 2), para 8 bits e posto 30

A tabela abaixo contém os erros médios quadráticos para todos os postos com 8 bits de precisão CORDIC e com a precisão do Matlab:

Tabela 5.8: Erro médio quadrático com precisão CORDIC de 8 bits e precisão do Matlab para diferentes postos reduzidos.

tr[$\mathbf{Q}_{xx}$] - Redução com 8 Bits de Precisão			
Posto	Durante o CORDIC	Após o CORDIC	Precisão Matlab
64	6,1094	6,3892	0,0000
50	174,8167	1,6097	1,0911
45	31,8915	0,3596	2,9688
40	16,2390	3,3765	5,8388
35	20,9859	7,9376	10,2360
30	12,7952	13,2225	15,2539
25	18,4062	20,2005	18,3496
20	289,1904	25,8631	24,2888
15	27,1486	31,5385	32,8031

Assim como foi visto na Seção 5.6.4 para 8 bits de precisão, o vetor $\hat{\mathbf{x}}_{cordicred2}$ apresentou valores de MSE menores para posto 50, 45 e 40 do que o MSE com posto completo. Isto ocorreu porque, no caso de 8 bits de precisão, o erro de quantização dos últimos valores singulares foi retirado na redução destes postos, fazendo com que o MSE tenha ficado menor. A Tabela 5.8 mostra claramente este efeito.

Para o método 1, ao realizar o cálculo com a precisão de 1 bit, é introduzido um elevado erro de quantização antes da redução. Este MSE elevado vai diminuindo gradativamente à medida em que o posto vai sendo reduzido e por conseqüência o erro de quantização vai sendo retirado, até que o erro "bias-squared" supere este delta, como podemos observar na primeira coluna da Tabela 5.8. Isto torna a redução de posto para o método 1 imprecisa se a quantização baixar de 8 bits de precisão.

5.7 Resultados da aplicação

Neste capítulo, foram vistos os efeitos do filtro de Wiener com posto reduzido sobre a matriz de autocorrelação $\mathbf{R}_{yy}$ através da SVD_CORDIC e SVD_CORDICRED, com precisão variável.

Os resultados apresentados para os cinco filtros de Wiener nas coordenadas padrão (posto completo e precisão do Matlab, posto reduzido e precisão do Matlab, posto completo e precisão dada pela SVD_CORDIC, posto reduzido após o cálculo da precisão pela SVD_CORDIC, e posto reduzido em conjunto com o cálculo dos bits de precisão) apresentaram o comportamento dentro do esperado.

É importante ressaltar o efeito do erro de quantização sobre o erro "*bias-squared*", que faz com que a redução de posto em alguns casos apresente um menor MSE do que o posto completo com precisão finita. Assim como o fato do primeiro método de redução de posto, cujo cálculo se dá após uma iteração CORDIC, apresentar um MSE maior do que o método de reduzir o posto após a determinação da precisão para a aplicação de estimação.

Outro ponto relevante é que apesar da complexidade assintótica da SVD_CORDIC ser a mesma complexidade assintótica da SVD de Jacobi, aumentamos o número de flops consideravelmente no Matlab. Isto também era esperado, pois o número de flops pelo método de Jacobi é determinado pela tolerância que se admite para diagonalizar a matriz desejada. O tempo computacional das funções desenvolvidas nesta tese, se compiladas num processador CORDIC bidimensional (com somadores e registradores de deslocamento) seria mais rápido do que o tempo apresentado pelo Matlab, independentemente do número de flops calculados. Para exemplificar, podemos citar Van Loan [1] quando afirma que a multiplicação de uma matriz triangular não poderia ser seis vezes mais rápida do que a multiplicação de uma matriz quadrada, concluindo que o método de contagem dos flops só captura uma das muitas dimensões que a eficiência de um algoritmo apresenta.

Capítulo 6

Conclusão

6.1 Contribuições da Tese

O principal objetivo e motivação da tese foi a busca de um método para redução de posto com precisão variável que pudesse ser aplicado em diversas áreas de processamento de sinais e comunicações móveis.

Propusemos dois métodos para obter o posto e a precisão reduzidos:

1. Um método que reduz o posto após a escolha da precisão para n bits, dada pela função SVD_CORDIC;

2. Um método que realiza a primeira iteração CORDIC, reduz o posto e depois continua a calcular a precisão desejada para mais *n-1* iterações CORDIC.

Estes dois métodos utilizam o algoritmo de Jacobi para fazer a decomposição em valores singulares. A idéia é diagonalizar a matriz desejada através de rotações de vetores bidimensionais, mantendo constante o valor da norma de Frobenius. A precisão finita CORDIC é aplicada diretamente na matriz de Givens, respeitando as suas condições de convergência e controlando a acurácia do algoritmo. Não aplicamos o CORDIC em todas as operações para evitar possíveis erros de arredondamento que não poderiam ser desprezados, e também para não precisar fazer um controle rígido sobre a convergência de cada operação.

Os resultados que obtivemos em cada simulação no Matlab estão comentados a seguir, subdivididos por capítulo:

- **Capítulo 2:** As funções CORDIC implementadas no Matlab (16 funções ao todo) e suas respectivas regiões de convergência apresentaram os resultados desejados. Poderíamos ter obtidos resultados mais expressivos nas simulações deste capítulo se a região de convergência CORDIC para o sistema de coordenadas linear e hiperbólico fosse expandida, através das seqüências de deslocamento. A pesquisa realizada neste capítulo criou uma biblioteca de funções CORDIC que podem ser utilizadas para obter precisão de n bits em outros algoritmos, cujas implementações também sejam feitas no Matlab.

- **Capítulo 3:** Implementamos a SVD pelo método de Jacobi no Matlab, fazendo testes com matrizes simétricas 3×3 e 5×5. As simulações foram bem sucedidas, mas o método de Jacobi limitou o uso dos métodos para reduzir o posto e a precisão em matrizes simétricas. Apesar de sua baixa complexidade, esta limitação nos impediu de obter análise dos resultados em outro domínio de coordenadas, como por exemplo obtenção do filtro de Wiener nas coordenadas de coerência e canônicas.

- **Capítulo 4:** Nesse capítulo foram implementados os métodos de posto e precisão reduzidos usando o CORDIC no algoritmo SVD, principais motivações originais da tese. A implementação foi feita com o auxílio do método de Jacobi. O exemplo deste capítulo mostra a correlação entre os dois métodos, que apresentam resultados semelhantes. Podemos inferir que o método proposto SVD_CORDICRED reduz a complexidade assintótica da SVD para $O(nr^2)$, sendo n as dimensões da matriz e r a nova dimensão após a redução de posto (visto na seção 4.4). Os métodos propõem uma alternativa de pesquisa na área de redução de posto podendo ser aprimorados e adaptados para mais ou menos bits de precisão com maior ou menor redução de posto. Por exemplo, pode-se adaptar o método SVD_CORDICRED para uma melhor relação entre posto $\times$ precisão, ou seja, após i iterações CORDIC, reduzir o posto e completar a precisão com n - i iterações. Outra forma de aprimorar o algoritmo é fazer uma iteração CORDIC, reduzir o posto em uma unidade e assim por diante, até se chegar à precisão ou ao posto desejados.

- **Capítulo 5:** Este capítulo junta todas as ferramentas pesquisadas neste tra-

balho, testadas numa aplicação de estimação. Foi implementado no Matlab um filtro de Wiener nas coordenadas padrão para que a matriz de autocorrelação $\mathbf{R}_{yy}$ pudesse ser decomposta pela SVD com precisão CORDIC. Foi feita a simulação para posto completo com precisão Matlab, posto reduzido com precisão Matlab, posto completo com precisão dada pela SVD_CORDIC, posto reduzido após a escolha da precisão pela SVD_CORDIC, e posto reduzido ao mesmo tempo em que a SVD_CORDICRED calcula os resultados com a precisão desejada. Podemos inferir a partir dos resultados apresentados o funcionamento dos métodos propostos numa aplicação de estimação, mas os resultados poderiam ser melhorados caso o filtro de Wiener no sistema de coordenadas padrão fosse implementado no sistema de coordenadas canônicas ou de coerência, como sugere Scharf [17].

Este trabalho deixa como contribuição acadêmica um tutorial sobre Cordic e suas aplicações em operações aritméticas, uma biblioteca CORDIC para ser usada no Matlab, uma análise sobre o filtro de Wiener utilizando a SVD_CORDIC e dois métodos distintos para realizar a redução de posto de matrizes simétricas, sendo uma conseqüência da tese o aprofundamento destes métodos para o uso em outras aplicações, como por exemplo sistemas de comunicação multiusuários.

Uma conclusão do Capítulo 5 foi o fato do MSE melhorar com a redução de posto para precisões baixas, aferido pelas simulações com o filtro de Wiener. A explicação para este efeito independe do método proposto, e se baseia nos efeitos da redução de posto. Quando o posto é reduzido, é retirado uma certa quantidade de informação do sistema e seus erros de quantização inerentes. Quando estes erros começam a crescer devido ao uso de menos bits para quantização, a redução de posto retira esses erros indesejáveis em maior quantidade do que a própria informação, como podemos ver na Tabela 5.5. Ou seja, vale à pena reduzir o posto e a precisão para diminuir a complexidade de algoritmos que possuem matrizes de ordem elevada em pró de um desempenho desejado. Apesar do aumento do número de flops pela decomposição em valores singulares, há de se ressaltar que a contagem de flops pode não refletir com exatidão a eficiência de um algoritmo.

6.2 Propostas de Trabalhos Futuros

É proposto como um trabalho futuro a implementação da SVD por outro método que não o de Jacobi para comparação com os resultados obtidos nesta tese, tendo como conseqüência a decomposição SVD de qualquer matriz $m \times n$.

A implementação do filtro de Wiener nas coordenadas canônincas e de coerência com o uso da SVD_CORDIC para matrizes $n \times m$ possibilitaria uma melhora nos resultados obtidos, e sua comparação com o trabalho de Scharf.

Outro passo sugerido para trabalhos futuros é a modificação dos métodos de posto e precisão reduzidos adaptados para novas relações posto x precisão, como falamos anteriormente na seção 6.1.

Referências Bibliográficas

[1] GOLUB, G. H., LOAN, C. F. V., *Matrix Computations*. The Johns Hopkins University Press, 1996.

[2] VOLDER, J. E., "The CORDIC trigonometric computing technique", *IRE Trans. Electronic Computers*, v. EC-8, n. 3, pp. 330–34, September 1959.

[3] WALTHER, J. S., "A unified algorithm for elementary functions", *AFIPS Spring Joint Computer Conference*, v. 38, pp. 379–85, 1971.

[4] MEGGITT, J. E., "Pseudo Division and Pseudo Multiplication Processes", *IBM J.*, pp. 210–226, April 1962.

[5] DAGGETT, D. H., "Decimal-Bonary Conversions in CORDIC", *IEEE Trans. on Electronic Computers*, v. EC-8, n. 3, pp. 335–39, September 1959.

[6] HSIAO, S. F., DELOSME, J. M., "Householder CORDIC Algorithms", *IEEE Transactions on Computers*, v. C-44, n. 8, pp. 990–1001, August 1995.

[7] ANDRAKA, R., "A survey of CORDIC algorithms for FPGA based computers", *ACM*, pp. 191–200, February 1998.

[8] HU, Y. H., "The Quantization Effects of the CORDIC Algorithm", *IEEE Transactions on Signal Processing*, v. 40, pp. 834–844, July 1992.

[9] DAWID, H., MEYR, H., *CORDIC Algorithms and Architectures*. Signal Processing Series, Marcel Dekker Inc., February 1999.

[10] HWANG, K., *Computer Arithmetic: Principles, Architectures, and Design*. John Wiley and Sons, 1979.

[11] HAVILAND, G. L., TUSZYNSKI, A. A., "A CORDIC arithmetic Processor chip", *IEEE Transactions on Computers*, v. C-29, n. 2, pp. 68–79, February 1980.

[12] MEHLING, R., MEYER, R., "CORDIC-AU, a suitable supplementary Unit to a General-Purpose Signal Processor", *AEÜ*, v. 43, n. 6, pp. 394–397, 1989.

[13] STRANG, G., *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 1988.

[14] HAYKIN, S., *Modern Filters*. Macmillan Coll Div, 1989.

[15] DINIZ, P. S. R., *Adaptive Filtering*. Kluwer Academic Publishers, 2002.

[16] DINIZ, P. S. R., SILVA, E. A. B. D., NETTO, S. L., *Digital Signal Processing*. Cambridge University Press, 2002.

[17] SCHARF, L. L., THOMAS, J., "Wiener Filters in Canonical Coordinates for Transform Coding, Filtering and Quantizing", *IEEE Transactions on Signal Processing*, v. 46, n. 3, pp. 647–654, March 1998.