UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA ELÉTRICA

MÉTODOS DE IMPLEMENTAÇÃO DE FUNÇÕES NÃO LINEARES EM FPGA

TIAGO DE CASTRO FALCÃO GUIMARÃES

Juiz de Fora - MG, Agosto de 2025

TIAGO DE CASTRO FALCÃO GUIMARÃES

MÉTODOS DE IMPLEMENTAÇÃO DE FUNÇÕES NÃO LINEARES EM FPGA

Trabalho de Conclusão de Curso apresentado junto ao programa de **Graduação em Engenharia Elétrica - Sistemas Eletrônicos** da **Universidade Federal de Juiz de Fora**.

Orientador:

Prof. Dr. Luciano Manhães de Andrade Filho.

Juiz de Fora - MG, Agosto de 2025

TIAGO DE CASTRO FALCÃO GUIMARÃES

MÉTODOS DE IMPLEMENTAÇÃO DE FUNÇÕES NÃO LINEARES EM FPGA

Trabalho de conclusão de curso apresentado ao programa de **Graduação em Engenharia Elétrica - Sistemas Eletrônicos** da **Universidade Federal de Juiz de Fora**.
Data de aprovação:
22/08/2025

Banca Examinadora:

Prof. Dr. Luciano Manhães de Andrade Filho
Universidade Federal de Juiz de Fora

Prof. Dr. Eder Barboza Kapisch
Universidade Federal de Juiz de Fora

Me. Thiago Campos Acácio Paschoalin
Universidade Federal de Juiz de Fora

Dedico esse trabalho aos meus pais, aos meus parentes, à minha namorada e aos meus amigos.

"O gasto mais custoso que se pode fazer é o do tempo."

Teofrasto, atribuído a Teofrasto, fonte incerta

RESUMO

Este trabalho apresenta a implementação e comparação de diferentes métodos de aproximação para funções não lineares em um processador *soft-core* de arquitetura reduzida, o *SAPHO*. São analisadas as funções seno, cosseno, tangente, arco tangente, exponencial, tangente hiperbólica e sigmoide, utilizando três métodos distintos: séries de Maclaurin, tabelamento e o algoritmo *CORDIC*. As implementações foram realizadas em duas configurações distintas de largura de dados, 23 e 32 *bits*. Os resultados foram avaliados com base no erro médio e no número de ciclos de clock exigidos para a execução. O estudo mostra que o uso de séries de Maclaurin tende a apresentar maior precisão, especialmente em 32 bits, ao custo de maior tempo de execução. O tabelamento apresenta bom equilíbrio entre precisão e desempenho. O algoritmo CORDIC, por sua vez, mostrou desempenho inferior no *SAPHO* por necessitar de alta precisão e tratamento de erros nos circuitos que realizam cálculo em ponto-flutuante. As análises são complementadas com gráficos, tabelas e trechos de código que sustentam as conclusões obtidas.

Palavras-chave: FPGA. Aproximações Numéricas. Funções Não Lineares. Processador Embarcado.

ABSTRACT

This work presents the implementation and comparison of different approximation methods for nonlinear functions in a reduced-architecture soft-core processor, the *SAPHO*. The functions analyzed include sine, cosine, tangent, arctangent, exponential, hyperbolic tangent, and sigmoid, using three distinct methods: Maclaurin series, lookup tables, and the *CORDIC* algorithm. Implementations were carried out using two different data widths, 23 and 32 bits. The results were evaluated based on the mean error and the number of clock cycles required for execution. The study shows that the use of Maclaurin series tends to provide higher precision, especially in 32-bit configurations, at the cost of increased execution time. Lookup tables offer a good balance between accuracy and performance. The *CORDIC* algorithm using floating point data, in turn, showed lower performance on the *SAPHO*. The analyses are supported by graphs, tables, and code excerpts that reinforce the conclusions presented.

Keywords: FPGA. Numerical Approximations. Nonlinear Functions. Embedded Processor.

LISTA DE FIGURAS

Figura 2.1 – Diagrama de blocos da arquitetura do processador *SAPHO* (VICCINI, 2023). 18
Figura 2.2 – Interface principal da IDE do SAPHO, com as cinco áreas destacadas. . 20
Figura 2.3 – Tela de configuração de um novo processador na IDE do SAPHO. . . . 21
Figura 4.1 – Simetrias da função seno. 33
Figura 4.2 – Simetrias da função tangente. 34
Figura 4.3 – Simetrias da função arco tangente. 35
Figura 6.1 – Erro da função seno através de séries de Maclaurin. 48
Figura 6.2 – Erro da função cosseno através de séries de Maclaurin. 49
Figura 6.3 – Erro da função tangente através de séries de Maclaurin. 49
Figura 6.4 – Erro da função tangente através de séries de Maclaurin Ampliada. . . . 50
Figura 6.5 – Erro da função arco tangente através de séries de Maclaurin. 50
Figura 6.6 – Erro da função exponencial através de séries de Maclaurin. 51
Figura 6.7 – Erro da função tangente hiperbólica através de séries de Maclaurin. . . 52
Figura 6.8 – Erro da função sigmoid através de séries de Maclaurin. 52
Figura 6.9 – Erro da função seno através de tabelamento. 54
Figura 6.10–Erro da função cosseno através de tabelamento. 55
Figura 6.11–Erro da função tangente através de tabelamento. 55
Figura 6.12–Erro da função tangente através de tabelamento ampliado. 56
Figura 6.13–Erro da função arco tangente através de tabelamento. 57
Figura 6.14–Erro da função tangente hiperbólica através de tabelamento. 57
Figura 6.15–Erro da função sigmoide através de tabelamento. 58
Figura 6.16–Erro da função seno através do algoritmo CORDIC. 59
Figura 6.17–Erro da função seno através do algoritmo CORDIC com 32 bits. 59
Figura 6.18–Erro da função cosseno através do algoritmo CORDIC. 61
Figura 6.19–Erro da função Cosseno através do algoritmo CORDIC com 32 bits. . . 61
Figura 6.20–Erro da função tangente através do algoritmo CORDIC. 62
Figura 6.21–Erro da função tangente através do algoritmo CORDIC ampliada. . . . 62
Figura 6.22–Erro da função arco tangente através do algoritmo CORDIC. 63
Figura 6.23–Erro da função exponencial através do algoritmo CORDIC. 64
Figura 6.24–Erro da função tangente hiperbólica através do algoritmo CORDIC. . . 64
Figura 6.25–Erro da função sigmoide através do algoritmo CORDIC. 65
Figura 7.1 – Gráfico de barras do erro médio de cada método apresentado. 68
Figura 7.2 – Gráfico de barras do número de *clocks* médio de cada método apresentado. 69

LISTA DE TABELAS

Tabela 2.1 – Instruções e respectivos circuitos criados automaticamente na *ULA* (VICCINI, 2023). 19
Tabela 6.1 – Resumo dos intervalos de tabelamento utilizados. 53
Tabela 6.2 – Erro médio com séries de Maclaurin. 65
Tabela 6.3 – Número de ciclos de clock médio com séries de Maclaurin. 65
Tabela 6.4 – Erro médio com tabelamento. 66
Tabela 6.5 – Número de ciclos de clock médio com tabelamento. 66
Tabela 6.6 – Erro médio com *CORDIC*. 66
Tabela 6.7 – Número de ciclos de clock médio com *CORDIC*. 67

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
TCC	Trabalho de Conclusão do Curso
NBR	Norma Brasileira
SAPHO	Scalable Architecture Processor for Hardware Optimization
PSC	Processador Soft-Core
FPGA	Field Programmable Gate Arrays
CORDIC	Coordinate Rotation Digital Computer
ULA	Unidade Lógica e Aritmética
IEEE	Institute of Electrical and Electronics Engineers

LISTA DE SÍMBOLOS

$f(x)$	Função de x
$f'(x)$	Derivada da função de x
$f''(x)$	Derivada de segunda ordem da função de x
$f'''(x)$	Derivada de terceira ordem da função de x
$f^{(n)}(x)$	Derivada de n-ésima ordem da função de x
$\approx$	Aproximação
$\sum$	Somatório
$\in$	Pertence
π	Pi
$\frac{d}{dx}$	Derivada da função em relação à x
$\binom{m}{k}$	Binômio de m k
$\|x\|$	Valor absoluto de x
$>$	Maior que
$<$	Menor que
$\pm$	Mais ou menos
∞	Infinito
ϕ	Phi
$\prod$	Produtório
$\neq$	Diferente de
σ	Sigma

SUMÁRIO

1 INTRODUÇÃO 16

2 O PROCESSADOR SAPHO 18

2.1 Arquitetura do Processador 18

2.2 Software 20

2.3 Funcionamento e Fluxo de Dados 22

3 IMPLEMENTAÇÃO POR MEIO DE SÉRIES DE MACLAURIN 23

3.1 Seno 24

3.2 Cosseno 25

3.3 Tangente 26

3.4 Arco Tangente 27

3.5 Exponencial 29

3.6 Tangente Hiperbólica 30

3.7 Sigmoide 31

4 TABELAMENTO 32

4.1 Fundamentação Teórica 32

4.2 Seno 32

4.3 Cosseno 33

4.4 Tangente 33

4.5 Arco Tangente 34

4.6 Exponencial 35

4.7 Tangente Hiperbólica 36

4.8 Sigmoide 36

5 ALGORITMO CORDIC 37

5.1 Modo Circular 37

5.2 Modo Vetorial 40
5.3 Modo Hiperbólico 41

6 IMPLEMENTAÇÃO EM CÓDIGO E COMPARAÇÃO DOS RESULTADOS 46
6.1 Maclaurin 47
6.1.1 Seno, Cosseno e Tangente 47
6.1.2 Arco Tangente, Exponencial, Tangente Hiperbólica e Sigmoide 48
6.2 Tabelamento 52
6.2.1 Seno, Cosseno e Tangente 53
6.2.2 Arco Tangente, Tangente Hiperbólica e Sigmoide 56
6.3 CORDIC 56
6.3.1 Seno, Cosseno e Tangente 58
6.3.2 Arco Tangente, Exponencial, Tangente Hiperbólica e Sigmoide 63
6.4 Comparação dos Resultados 65

7 CONCLUSÃO 68

REFERÊNCIAS 70

1 INTRODUÇÃO

O *Scalable Architecture Processor for Hardware Optimization*, ou *SAPHO*, é um *Processador Soft-Core PSC* desenvolvido no Núcleo de Instrumentação e Processamento de Sinais da Universidade Federal de Juiz de Fora. Diferentemente de outras arquiteturas de *PSCs*, o *SAPHO* permite a alocação automática de recursos de hardware durante a compilação, gerando circuitos otimizados conforme as instruções presentes no código do usuário. Por ser utilizado em *Field Programmable Gate Arrays*, *FPGAs*, o *SAPHO* é capaz de realizar o processamento de um grande volume de dados de maneira rápida, precisa e com baixa utilização de recursos computacionais (VICCINI, 2023).

Na busca em abranger uma maior gama de utilização e aumentar a capacidade de processamento, é necessário expandir a biblioteca em C de funções do processador, para aproximar certas funções não-lineares, buscando uma boa relação entre precisão e velocidade. Uma alternativa possível é utilizar as bibliotecas padrão, que calculam os resultados de funções matemáticas em C, e adaptá-las às particularidades do processador. Porém, em sistemas embarcados com recursos restritos, a utilização das funções contidas na biblioteca *math.h* não é computacionalmente adequada, pois estas dependem de suporte de hardware específico, como *floating point unit* padronizada, e exigem tabelas e lógica complexa para tratar casos especiais, e não oferecem controle sobre os ciclos de execução e parâmetros para a parada das estimativas. Assim, ao desenvolver as funções não lineares nativas, é possível gerar implementações portáveis, com precisão configurável e ciclos determinísticos, além de reduzir drasticamente o tamanho do código, que são fatores fundamentais em aplicações embarcadas (BRIGGS I. LAD, 2023). Sendo assim, foi necessário buscar outros meios para aproximar as funções de interesse.

A alternativa apresentada neste trabalho é a criação e comparação de métodos distintos de se aproximar as funções seno, cosseno, tangente, arco tangente, exponencial, tangente hiperbólica e sigmoide. Foram empregadas séries de Maclaurin, tabelas de consulta com interpolação e o algoritmo *Coordinate Rotation Digital Computer*, ou *CORDIC*, cada um com dois conjuntos de parâmetros distintos do processador. Para isso, partiu-se do desenvolvimento de um conjunto de códigos em *C* que implementa as funções relevantes. Esse código foi inicialmente adaptado para a linguagem de programação do *SAPHO*, onde posteriormente foi testado, tanto em termos de precisão quanto em ciclos de *clock*, para cada método em cada configuração do processador.

Desta forma, esse trabalho foi estruturado da seguinte maneira: o Capítulo 2 explica o funcionamento e particularidades do processador *SAPHO*. O Capítulo 3 apresenta a base

teórica utilizada para a implementação de séries de Maclaurin em código. O Capítulo 4 explica como foi realizado a utilização de tabelas para as comparações feitas. O Capítulo 5 apresenta a fundamentação teórica por trás dos códigos que utilizam o algoritmo *CORDIC*. O Capítulo 6 apresenta os comportamentos vistos para cada um dos 40 métodos distintos empregados. E o Capítulo 7 compara os resultados apresentados no Capítulo 6.

2 O PROCESSADOR SAPHO

Neste capítulo, é apresentado o processador *SAPHO*, com foco em seus aspectos arquiteturais, fluxo de dados e funcionamento interno. A proposta do *SAPHO*, isto é *Scalable Architecture Processor for Hardware Optimization*, é oferecer alto desempenho e flexibilidade para aplicações de processamento intensivo (KAPISCH E.B.,).

O processador *SAPHO* foi criado em um contexto da busca por soluções que otimizem a eficiência na execução de tarefas computacionais complexas. Seu desenvolvimento foi orientado para atender demandas de alto desempenho em ambientes com hardwares distintos, com ênfase em modularidade e reconfigurabilidade (VICCINI, 2023).

2.1 ARQUITETURA DO PROCESSADOR

A arquitetura do *SAPHO* foi concebida para oferecer uma alta taxa de processamento aliado a uma exigência de recursos computacionais relativamente baixa (VICCINI, 2023). Essa arquitetura modular permite escalabilidade e a integração de diversas unidades funcionais especializadas. A Figura 2.1 ilustra, de forma esquemática, a estrutura modular do processador, ressaltando os principais blocos de controle e dados.

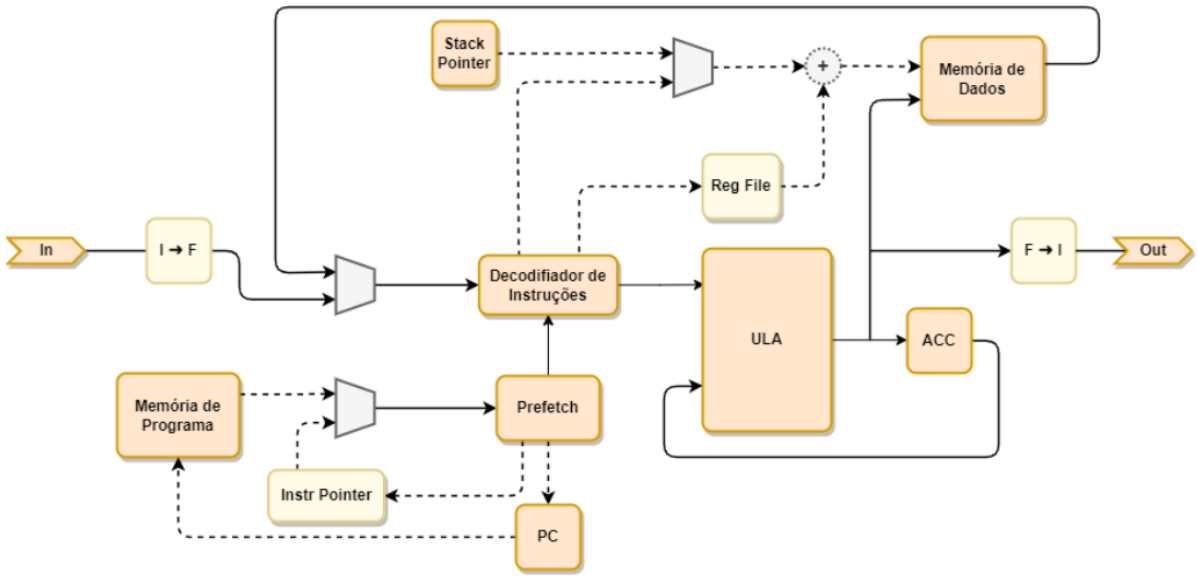

Figura 2.1 – Diagrama de blocos da arquitetura do processador *SAPHO* (VICCINI, 2023).

As unidades funcionais do *SAPHO* são responsáveis pela execução das instruções e manipulação dos dados. Dentre elas, destacam-se (KAPISCH E.B.,):

- **Unidade de Controle:** Gerencia o fluxo de instruções e coordena as operações entre os diversos módulos, garantindo que os sinais de controle sejam enviados no tempo adequado.
- **Unidade Lógica e Aritmética (*ULA*):** Responsável pelo processamento de operações matemáticas e lógicas, a *ULA* foi otimizada para realizar operações de forma paralela sempre que possível.
- **Unidade de Memória:** Encapsula os mecanismos de acesso à memória, implementando políticas de cache e gerenciamento que aumentam a eficiência no processamento dos dados.

O *Register File* é gerado sempre que há utilização de arrays no programa do usuário e tem como função indexar corretamente os elementos do vetor. Ele faz offset na posição de memória do primeiro elemento para acessar o elemento desejado de forma eficiente e sem lógica adicional em software (VICCINI, 2023).

A *ULA* do *SAPHO* possui grau significativo de flexibilidade, parametrizada automaticamente com base nas instruções geradas pelo compilador Assembly e em parâmetros de projeto, como ponto flutuante ou ponto fixo, tamanho da palavra em bits, selecionados pelo usuário. A Tabela 2.1 apresenta os circuitos criados dentro da ULA, de acordo com cada instrução.

Tabela 2.1 – Instruções e respectivos circuitos criados automaticamente na *ULA* (VICCINI, 2023).

Instrução	Circuito na *ULA*	Ponto Fixo	Ponto Flutuante
DIV	X	X	X
OR	X	X	
LOR	X	X	X
GRE	X	X	X
MOD	X	X	
MLT	X	X	X
LES	X	X	X
EQU	X	X	X
AND	X	X	
LAN	X	X	X
INV	X	X	
LIN	X	X	X
SHR	X	X	
SHL	X	X	
SRS	X	X	
CALL	X	X	
SRF	X	X	

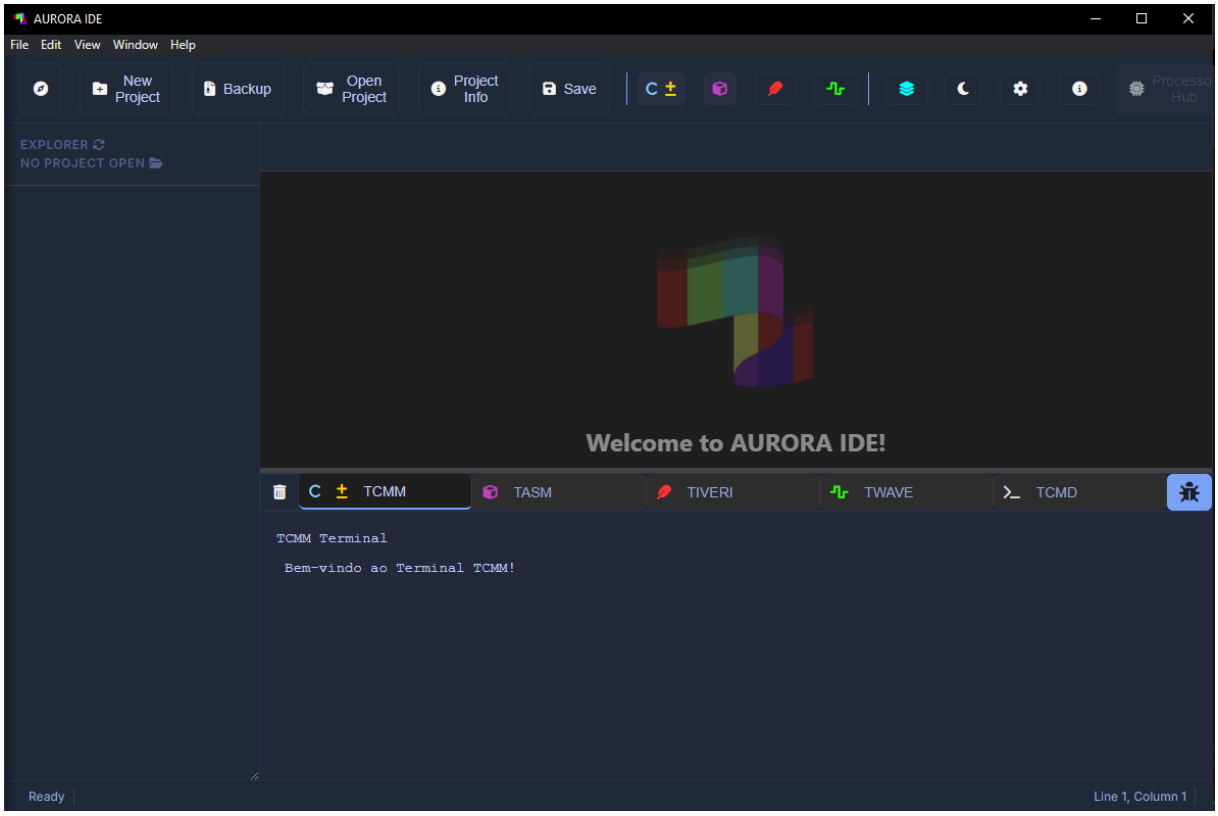

Figura 2.2 – Interface principal da IDE do SAPHO, com as cinco áreas destacadas.

2.2 SOFTWARE

A geração automática do código Verilog e dos arquivos de inicialização de memória é feita a partir do código utilizando uma IDE dedicada para compilar a sublinguagem de C e Assembly utilizados no processador(VICCINI, 2023).

A IDE do *SAPHO* foi implementada em C# e pode ser vista na Figura 2.2.

Após criar o projeto, é possível criar um processador e definir os parâmetros de sua *ULA*. Também é possível definir o número de portas de entrada e saída, o número de quantização de bits e os tamanhos das pilhas de dados e de instruções. A Figura 2.3 mostra a tela de configuração.

O compilador, desenvolvido com Flex e Bison, traduz um subconjunto da linguagem C, em Assembly e gera diretivas de compilação que definem a parametrização do hardware (VICCINI, 2023). O compilador Assembly reconhece 49 instruções e produz código Verilog parametrizado e arquivos `data.mif` e `inst.mif` para inicialização de memória. Ele utiliza Flex para identificar *opcodes* e operandos, dimensionando corretamente as memórias de programa e dados conforme o número de instruções e variáveis detectadas (VICCINI, 2023).

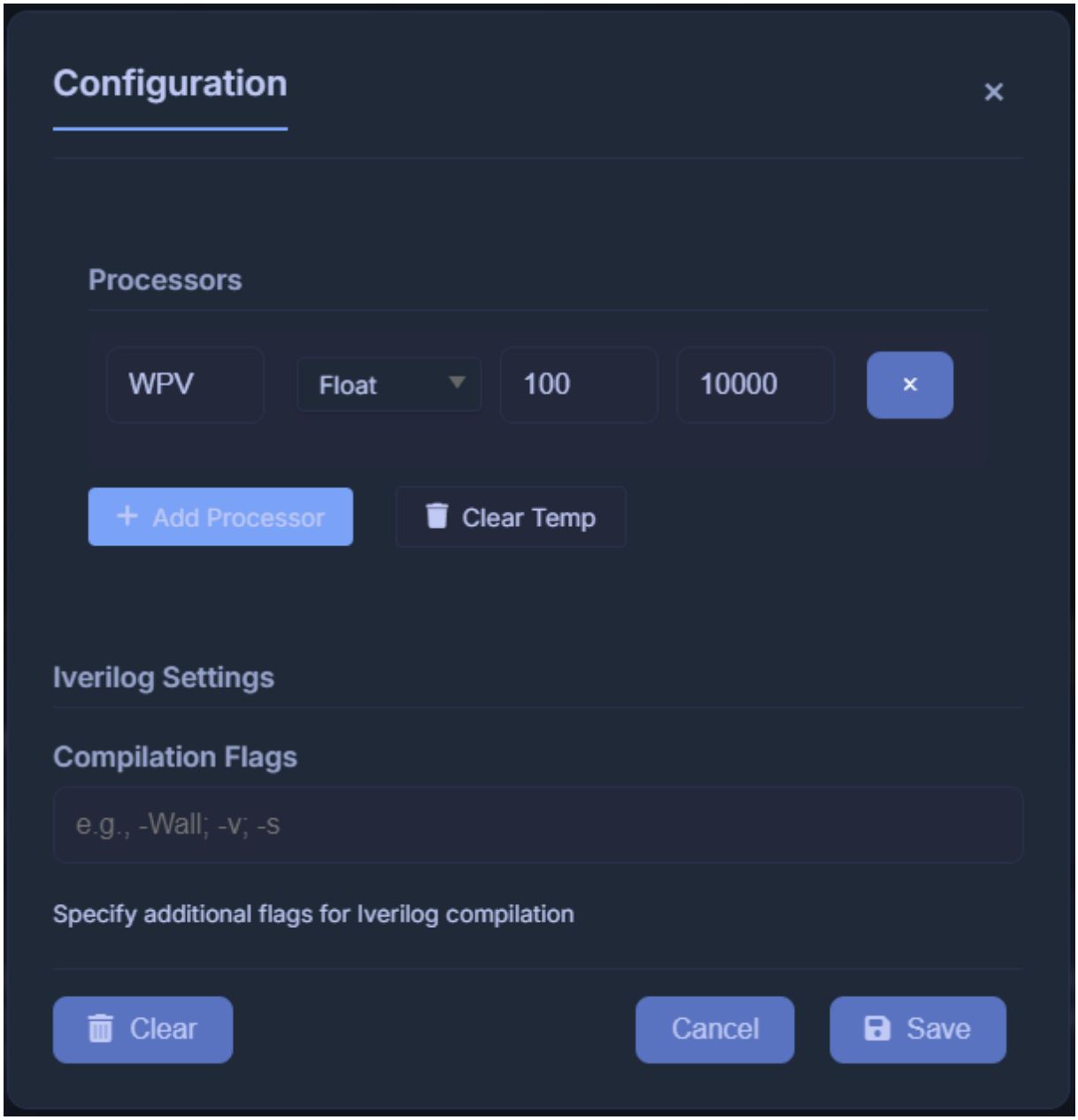

Figura 2.3 – Tela de configuração de um novo processador na IDE do SAPHO.

Embora o padrão IEEE 754 defina os formatos *single* de 32 bits e *double* de 64 bits, o *SAPHO* emprega representação simplificada com expoente em complemento de dois, reduzindo recursos de hardware e ciclos de clock. O número é representado como mostrado na Equação 1, onde S é o bit de sinal, M a mantissa normalizada em $[1, 2)$ e E o expoente em complemento de dois (VICCINI, 2023). Essa abordagem mantém a compatibilidade conceitual e otimiza o fluxo de dados em operações aritméticas.

$$(-1)^S \times M \times 2^E \tag{1}$$

2.3 FUNCIONAMENTO E FLUXO DE DADOS

O funcionamento do processador *SAPHO* é marcado pela integração entre hardware e algoritmos de controle. A execução das instruções inicia-se pela decodificação, seguida pela alocação dos recursos necessários para a operação. Durante o ciclo de processamento, os dados são encaminhados entre as unidades funcionais de forma orquestrada para que a computação ocorra com o mínimo de latência.

Os mecanismos de predição e verificação de dependências foram implementados para evitar conflitos e reduzir os tempos de espera entre os ciclos de execução. Segundo (VICCINI, 2023), essa abordagem possibilita melhorias significativas na eficiência do processamento, especialmente em ambientes com elevada concorrência de tarefas.

3 IMPLEMENTAÇÃO POR MEIO DE SÉRIES DE MACLAURIN

Este capítulo apresenta a fundamentação teórica e as estratégias de implementação baseadas em séries, utilizadas para aproximar funções não lineares no processador *SAPHO*. São abordadas as funções seno, cosseno, tangente, arco tangente, exponencial, tangente hiperbólica e sigmoide.

Uma possível ferramenta na busca de métodos computacionais para o cálculo de funções não lineares é a Série de Taylor. (STEIN E.M., 2003). Como método inicial para o cálculo das funções propostas, pode-se utilizar funções na linguagem de programação C para a realização de aproximações de Séries de Taylor. Segue o formato geral da Série de Taylor (BOYCE W.E., 2017):

$$f(x) = f(a) + \frac{f'(a)(x-a)}{1!} + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{2}$$

Em que:

- $f(a)$ é o valor da função no ponto de expansão;
- $f'(a), f''(a), \ldots, f^{(n)}(a)$ são, respectivamente, as derivadas de primeira, segunda e enésima ordem da função, avaliadas no ponto a;
- $n!$ representa o fatorial de n;

Tendo como objetivo tornar o cálculo menos computacionalmente custoso (PRESS W.H., 2002), é possível utilizar Séries de Maclaurin (STEIN E.M., 2003), que são Séries de Taylor com sua expansão centrada em zero, tendo o seguinte formato:

$$f(x) \approx f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots + \frac{f^{(n)}(0)}{n!}x^n \tag{3}$$

Após o cálculo da Série de Maclaurin, certas identidades trigonométricas podem ser aplicadas a fim de deixar o cálculo mais preciso e de convergência mais rápida (STEIN E.M., 2003).

3.1 SENO

Como exemplo da aplicação das séries no trabalho, considera-se a função seno, amplamente utilizada em aplicações que envolvem processamento de sinais, controle e modelagem matemática (GELFAND I.M., 2001).

Para que possa se expressar a Série de Taylor da função seno, é necessário ter as derivadas da função seno. Como $\frac{d}{dx}sin(x) = cos(x)$ e $\frac{d}{dx}cos(x) = -sin(x)$ (STEWART J.D., 2021), temos que as derivadas da função seno seguem um padrão:

$$\begin{aligned} f(x) &= \sin(x) \\ f'(x) &= \cos(x) \\ f''(x) &= -\sin(x) \\ f'''(x) &= -\cos(x) \\ f''''(x) &= \sin(x) \end{aligned}$$

É possível notar que as derivadas do seno seguem um ciclo de 4 derivadas, que posteriormente se repetem. Sendo assim, para o cálculo do seno, a série de Maclaurin é (GELFAND I.M., 2001):

$$\sin(x) \approx \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \tag{4}$$

Para a implementação em código que será apresentada no próximo capítulo do trabalho, utiliza-se a periodicidade da função seno no intervalo $x \in [0, 2\pi]$ (GELFAND I.M., 2001). Como as Séries de Taylor apresentam um erro crescente para valores mais distantes do ponto de expansão da série (STEIN E.M., 2003), no caso da série de Maclaurin a expansão é em torno de zero, torna-se necessário realizar as aproximações dentro de um intervalo próximo do valor 0. Assim, no código, valores fora do intervalo $[-\pi, \pi]$ são calculados com seus equivalentes pertencentes ao intervalo $x \in [0, 2\pi]$.

Para implementar a série mostrada na linguagem de Programação *C*, foi utilizada uma estrutura iterativa. Partindo do termo inicial não nulo, denotado aqui por T_1, que para a Série de Maclaurin da função seno é dada por $T_1 = x$, sendo x o ângulo a ser calculado pela série em radianos, tem-se a seguinte expressão para um determinado termo não nulo, com m representando um determinado índice ímpar:

$$T_m = (-1)^{(m-1)/2} \frac{x^m}{m!}, \quad m = 1, 3, 5, 7, 9, 11, ... \tag{5}$$

Sendo que o índice m assume apenas valores ímpares, no formato: $m = 1, 3, 5, 7, 9, 11....$ Enquanto para a relação entre dois termos consecutivos, uma vez que os termos pares da série de Maclaurin da função seno são nulos, T_m e à T_{m+2}:

$$\frac{T_{m+2}}{T_m} = \frac{(-1)^{(m+2)/2}\, x^{m+2}/(m+2)!}{(-1)^{m/2}\, x^m/m!} = \frac{-x^2}{(m+1)(m+2)} \tag{6}$$

Com T_m sendo um termo qualquer e T_{m+2} o próximo termo não nulo da série. Tendo a relação entre dois termos subsequentes da série, é possível calcular o valor de cada item da série de maneira iterativa, ao multiplicar o valor calculado para o termo anterior pelo valor determinado em 6 e somar esse valor aos demais. A operação é parada quando os termos calculados são menores em módulo do que uma tolerância pré-definida que será explorada posteriormente.

3.2 COSSENO

A função cosseno, assim como a função seno, apresenta um padrão cíclico em suas derivadas (STEWART J.D., 2021). Tendo em vista que, conforme apresentado na seção da função seno, a derivada de $\sin(x)$ é $\cos(x)$, é possível simplificar a análise da função cosseno utilizando essa relação intrínseca entre as duas funções (GELFAND I.M., 2001).

As derivadas da função $\cos(x)$ podem ser determinadas da seguinte forma:

$$\begin{aligned} f(x) &= \cos(x) \\ f'(x) &= -\sin(x) \\ f''(x) &= -\cos(x) \\ f'''(x) &= \sin(x) \\ f''''(x) &= \cos(x) \end{aligned}$$

Observa-se que, assim como no caso do seno, as derivadas do cosseno formam um ciclo com período 4 (STEWART J.D., 2021). Com as derivadas avaliadas em 0, obtém-se a Série de Maclaurin para $\cos(x)$:

$$\cos(x) \approx \sum_{n=0}^{N} (-1)^n \frac{x^{2n}}{(2n)!} \tag{7}$$

Assim como no caso do seno, onde se utiliza a periodicidade no intervalo $x \in [0, 2\pi]$ para garantir a precisão da aproximação, o mesmo procedimento é adotado para $\cos(x)$. Dessa forma, valores de x fora desse intervalo são ajustados para que a aproximação se mantenha dentro de uma região onde o erro da série de Maclaurin seja minimizado.

Sendo o termo inicial não nulo da Série de Maclaurin da função cosseno $T_0 = 1$, nota-se que o termo geral T_m da série assume o formato:

$$T_m \;=\; (-1)^{m/2}\frac{x^m}{m!}, \quad m = 0, 2, 4, 6, 8, 10, \ldots \tag{8}$$

A razão entre dois termos consecutivos não nulos (de índice m para $m + 2$) é, de forma análoga ao seno:

$$\frac{T_{m+2}}{T_m} \;=\; \frac{(-1)^{(m+2)/2}\, x^{m+2}/(m+2)!}{(-1)^{m/2}\, x^m/m!} \;=\; \frac{-x^2}{(m+1)(m+2)} \tag{9}$$

Com T_m representando o termo de índice par atual e T_{m+2} o termo subsequente não nulo, faz-se a iteração multiplicando o termo anterior pela razão em 9 e acumulando-se a soma à estimativa de cosseno de x. Antes de iniciar essa iteração, aplica-se a normalização do ângulo para que $|x|$ permaneça pequeno e o erro de truncamento da série seja minimizado. A iteração prossegue até que $|T_{m+2}|$ fique abaixo de uma tolerância definida conforme a precisão desejada em ponto flutuante.

3.3 TANGENTE

A função tangente é definida como a razão entre as funções seno e cosseno, conforme a expressão a seguir (GELFAND I.M., 2001):

$$\tan(x) = \frac{\sin(x)}{\cos(x)} \tag{10}$$

Ao realizar o procedimento de encontrar as derivadas sucessivas da função, da mesma forma de seno e cosseno, encontra-se os seguintes resultados (STEWART J.D., 2021):

$$
\begin{aligned}
f(x) &= \tan(x) \\
f'(x) &= \sec^2(x) \\
f''(x) &= 2\sec^2(x)\tan(x) \\
f'''(x) &= 2\sec^2(x)\left(\sec^2(x) + 2\tan^2(x)\right)
\end{aligned}
$$

É possível notar que não há um padrão cíclico nas derivadas da função tangente, diferentemente das funções seno e cosseno. Sendo assim, uma forma de obter o valor da tangente é calcular o valor do cosseno, do seno e então determinar sua razão, conforme a Equação 11.

$$
tan(x) \approx \frac{\sum_{n=0}^{N}(-1)^n \frac{x^{2n+1}}{(2n+1)!}}{\sum_{n=0}^{N}(-1)^n \frac{x^{2n}}{(2n)!}} \tag{11}
$$

Para se determinar o resultado de $tan(x)$ computacionalmente, esse método envolve o cálculo do valor aproximado da função $sen(x)$ e da função $cos(x)$.

3.4 ARCO TANGENTE

A abordagem para a implementação da função arco tangente utiliza um método um pouco diferente dos empregados nas subseções anteriores. A derivada da função arco tangente é dada por (STEIN E.M., 2003):

$$
\frac{d}{dx}\left(\arctan(x)\right) = \frac{1}{1+x^2} \tag{12}
$$

A partir da Série Geométrica, que segue o formato (STEIN E.M., 2003):

$$
\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots + x^n, \quad \text{para } x \in (-1, 1) \tag{13}
$$

Podemos reorganizar a equação 12 a partir da série geométrica (STEIN E.M., 2003):

$$
\frac{1}{1+x^2} = 1 - x^2 + x^4 - x^6 \cdots + (-1)^n x^{2n}, \quad \text{para } x \in (-1, 1) \tag{14}
$$

Ao integrar os dois lados da igualdade, temos (STEIN E.M., 2003):

$$arctan(x) = \int \frac{1}{1+x^2} dx = \int \left(1 - x^2 + x^4 - x^6 \cdots + (-1)^n x^{2n}\right) dx, \quad \text{para } x \in (-1, 1) \tag{15}$$

Por fim, temos que (STEIN E.M., 2003):

$$arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots + \frac{(-1)^n}{2n+1} x^{2n+1} \approx \sum_{n=0}^{N} \frac{(-1)^n}{2n+1} x^{2n+1}, \quad \text{para } x \in (-1, 1) \tag{16}$$

É importante enfatizar que a série acima converge apenas para $x \in (-1, 1)$. Para valores de x fora desse intervalo, torna-se necessário realizar ajustes que permitam reduzir o argumento da função para um valor no intervalo de convergência. Para isso, utiliza-se as seguintes identidades (GELFAND I.M., 2001):

$$\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right), \quad \text{se } x > 0 \tag{17}$$

$$\arctan(x) = -\frac{\pi}{2} - \arctan\left(\frac{1}{x}\right), \quad \text{se } x < 0 \tag{18}$$

Dessa forma, ao aplicar essas identidades, o valor de x é convertido para $\frac{1}{x}$, que pertence ao intervalo $(-1, 1)$, possibilitando a utilização da Série de Maclaurin para obter uma aproximação precisa de $\arctan(x)$. Na subsequente implementação do código, essas relações serão empregadas para garantir que a aproximação seja válida e precisa mesmo para argumentos fora do intervalo de convergência natural da série.

Com o argumento reduzido ao intervalo $(-1, 1)$, foram aplicadas também as identidades mostradas nas Equações 19 e 20, que permitem refinar a convergência da série ao restringir os cálculos ao intervalo $(-0{,}5, 0{,}5)$.

$$\arctan(x) = \arctan\left(\frac{x-1}{1+x}\right) + \frac{\pi}{4}, \quad \text{para } x > 0.5 \tag{19}$$

$$\arctan(x) = \arctan\left(\frac{x+1}{1-x}\right) - \frac{\pi}{4}, \quad \text{para } x < -0.5 \tag{20}$$

Como feito para seno e cosseno, define-se o termo inicial e a relação recursiva para os termos da série de Maclaurin da função arco tangente. O termo inicial não nulo é

$T_1 = x$. Para índices ímpares, isso é $m = 1, 3, 5, 7, 9, 11, \ldots$, o termo T_m pode ser escrito como:

$$T_m \;=\; (-1)^{\frac{m-1}{2}} \frac{x^m}{m} \tag{21}$$

Sendo assim, a razão entre termos consecutivos não nulos para um índice ímpar atual m e o próximo termo não nulo de índice $m + 2$ é:

$$\frac{T_{m+2}}{T_m} = \frac{(-1)^{\frac{(m+2)-1}{2}} \dfrac{x^{m+2}}{m+2}}{(-1)^{\frac{m-1}{2}} \dfrac{x^m}{m}} = \frac{-m}{m+2} x^2 \tag{22}$$

Assim, a cada passo multiplica-se o termo atual e acumula-se à soma parcial, até que $|T_{m+2}|$ fique abaixo da tolerância definida, encerrando a conta.

3.5 EXPONENCIAL

A função exponencial de Euler real, e^x, é de extrema importância para a realização de cálculos matemáticos para a engenharia (AGARWAL R.P, 2011). A sua expansão em Série de Maclaurin converge para todos os valores de x, o que a torna uma ferramenta poderosa para aproximações numéricas (BOYCE W.E., 2017).

A função e^x tem todas as suas derivadas de ordem n iguais à sua primitiva, isto é, e^x. Sendo assim, segundo (BOYCE W.E., 2017), a série de Maclaurin para e^x é dada por:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} \approx \sum_{n=0}^{N} \frac{x^n}{n!} \tag{23}$$

Tendo o termo inicial da série, $T_0 = 1$, é possível definir o valor do termo de índice m, com $m = 0, 1, 2, 3, 4, 5, \ldots$ a partir de:

$$T_m = \frac{x^m}{m!} \tag{24}$$

Sendo assim, a razão entre termos consecutivos T_m e T_{m+1} é:

$$\frac{T_{m+1}}{T_m} = \frac{x^{m+1}/(m+1)!}{x^m/m!} = \frac{x}{m+1} \tag{25}$$

O processo iterativo prossegue até que $|T_{m+1}|$ fique abaixo da tolerância definida, garantindo a precisão desejada.

Observa-se que, diferentemente da função arco tangente apresentada anteriormente, cuja série converge apenas para determinados valores de x, a série para e^x apresenta convergência global (AGARWAL R.P, 2011). Essa propriedade garante que, para qualquer valor de x, a aproximação pode ser aprimorada aumentando-se o número de termos N. Adicionalmente, vale ressaltar que a rápida convergência da série de e^x para valores próximos a 0 facilita a obtenção de resultados precisos com um número relativamente pequeno de termos, o que é vantajoso em ambientes computacionais com restrições de desempenho.

3.6 TANGENTE HIPERBÓLICA

A tangente hiperbólica é definida por (GELFAND I.M., 2001):

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{26}$$

Para facilitar os cálculos apresentados a seguir, a expressão acima pode ser organizada da seguinte forma:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{27}$$

Uma forma direta de se calcular o valor da tangente hiperbólica é utilizando a expansão em Série de Maclaurin para a exponencial real de $-2x$. Assim, temos:

$$e^{-2x} \approx \sum_{n=0}^{N} \frac{(-2x)^n}{n!} = \sum_{n=0}^{N} \frac{(-2)^n x^n}{n!}. \tag{28}$$

Substituindo a equação 28 na equação 26, a série aproximada para a Tangente Hiperbólica pode ser escrita como:

$$\tanh(x) \approx \frac{2}{1 + \sum_{n=0}^{N} \frac{(-2)^n x^n}{n!}} - 1. \tag{29}$$

De maneira similar ao caso anterior, nota-se que a série da tangente hiperbólica converge para qualquer valor de x, assim como a função exponencial real (STEIN E.M., 2003).

3.7 SIGMOIDE

A função sigmoide é definida como (KYURKCHIEV N., 2015):

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{30}$$

Aplicando a expansão em série da função exponencial, de maneira similar ao que foi realizado na equação (28), temos:

$$e^{-x} \approx \sum_{n=0}^{N} \frac{(-x)^n}{n!}, \tag{31}$$

De forma que a aproximação para a função sigmoide torna-se:

$$\sigma(x) \approx \frac{1}{1 + \sum_{n=0}^{N} \frac{(-x)^n}{n!}}. \tag{32}$$

Como a série da exponencial converge globalmente, este método é robusto para qualquer valor de x, embora a convergência seja mais rápida para valores próximos de zero (STEIN E.M., 2003).

4 TABELAMENTO

A ideia central do tabelamento é realizar uma comparação entre o valor que requer a aproximação e os valores guardados na memória em uma tabela. Ao pré-computar e armazenar os valores de uma função em intervalos definidos, dispensa-se o cálculo a cada chamada, o que é especialmente vantajoso em sistemas com recursos computacionais limitados (KANTABUTRA, 1996). Quando o valor requerido não está contido na tabela, uma aproximação é realizada utilizando técnicas de interpolação, sendo a interpolação linear a menos computacionalmente exigente e a que será empregada em código.

4.1 FUNDAMENTAÇÃO TEÓRICA

O tabelamento consiste em calcular e armazenar valores discretos de uma função em pontos determinados, x_i, onde $i = 0, 1, \ldots, N$. Quando o valor da função é requerido para um ponto x que não coincide exatamente com um dos pontos da tabela, utiliza-se um procedimento de interpolação para estimar $f(x)$ (KANTABUTRA, 1996). A interpolação linear é dada por (DRISCOLL T.A., 2018):

$$f(x) \approx f(x_i) + \left(\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \right) (x - x_i), \tag{33}$$

para $x \in [x_i, x_{i+1}]$.

Sendo assim, quando o valor solicitado não estiver contido diretamente na tabela, é realizada uma operação relativamente simples para a aproximação do valor desejado, utilizando poucos recursos computacionais (KANTABUTRA, 1996).

4.2 SENO

A função seno pode ser tabulada no intervalo $[0, \pi/2]$. Conforme demonstrado na Figura 4.1, é possível aproveitar as simetrias inerentes à função para reduzir o esforço computacional, restringindo o tabelamento a um intervalo menor (KANTABUTRA, 1996). Há relações de implementação computacional simples que podem ser aplicadas para encontrar a equivalência entre qualquer uma das quatro regiões apresentadas na Figura 4.1, para a região representada por *I*. Dessa forma, os valores da função seno para $x \in [0, 2\pi]$ podem ser obtidos a partir de transformações angulares apropriadas, mantendo a monotonicidade no intervalo selecionado (GELFAND I.M., 2001).

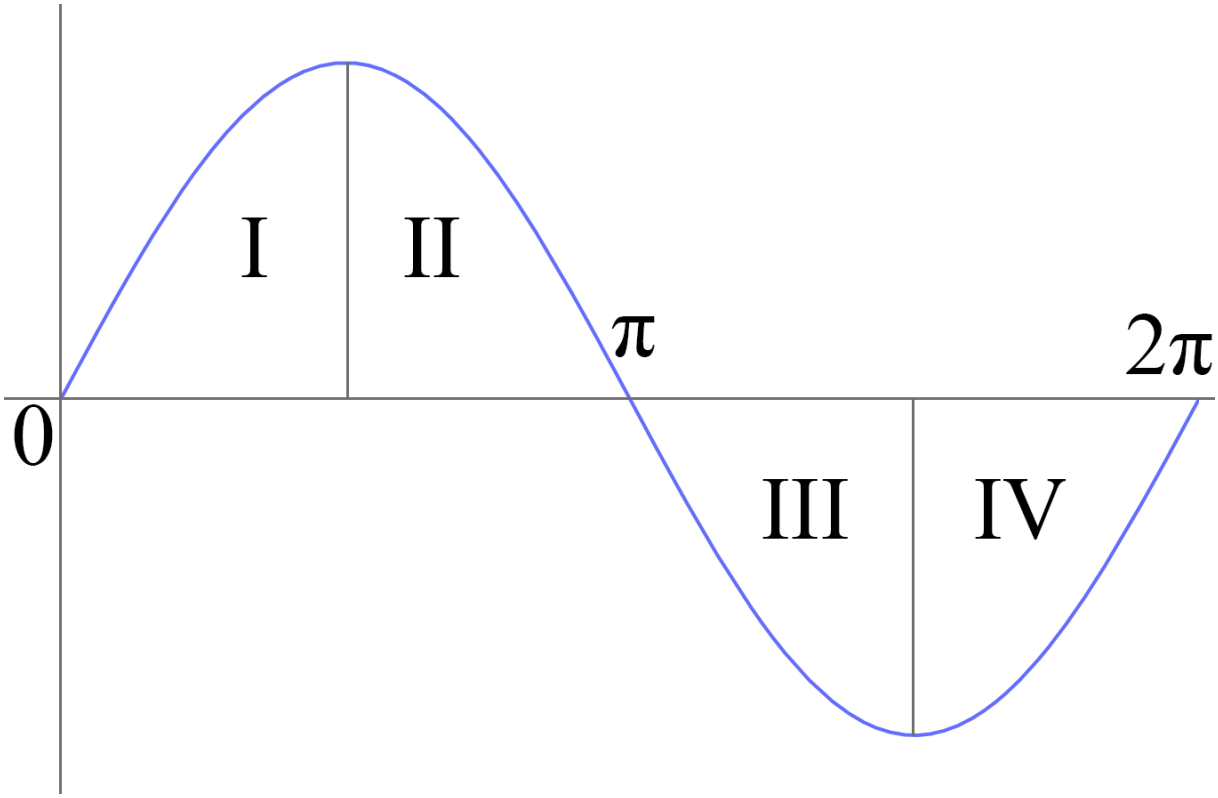

Figura 4.1 – Simetrias da função seno.

A interpolação linear é aplicada para aproximar os valores da função entre os pontos tabulados, o que facilita a obtenção de uma aproximação precisa sem a necessidade de armazenar um número elevado de valores, uma vez que, após ser enquadrado no intervalo $x \in [0, 2\pi]$, apenas os valores em $x \in [0, \pi/2]$ precisam ser armazenados.

4.3 COSSENO

A função cosseno também pode ser aproximada ao armazenar os seus valores no intervalo $[0, \pi/2]$, por razões similares ao caso da função seno (GELFAND I.M., 2001). Com base nas propriedades de simetria e na monotonicidade do intervalo escolhido, os valores da função para $x \in [0, 2\pi]$ podem ser recuperados por meio de transformações angulares dos pontos tabulados (GELFAND I.M., 2001). Para valores que não estão contidos nesse intervalo, é possível utilizar a periodicidade da função cosseno para permitir uma representação precisa (GELFAND I.M., 2001).

4.4 TANGENTE

Para a função tangente, a aproximação foi calculada a partir da razão das aproximações de seno e cosseno, uma vez que a função tangente apresenta descontinuidades em sua imagem (GELFAND I.M., 2001). Montar uma tabela para a função tangente de

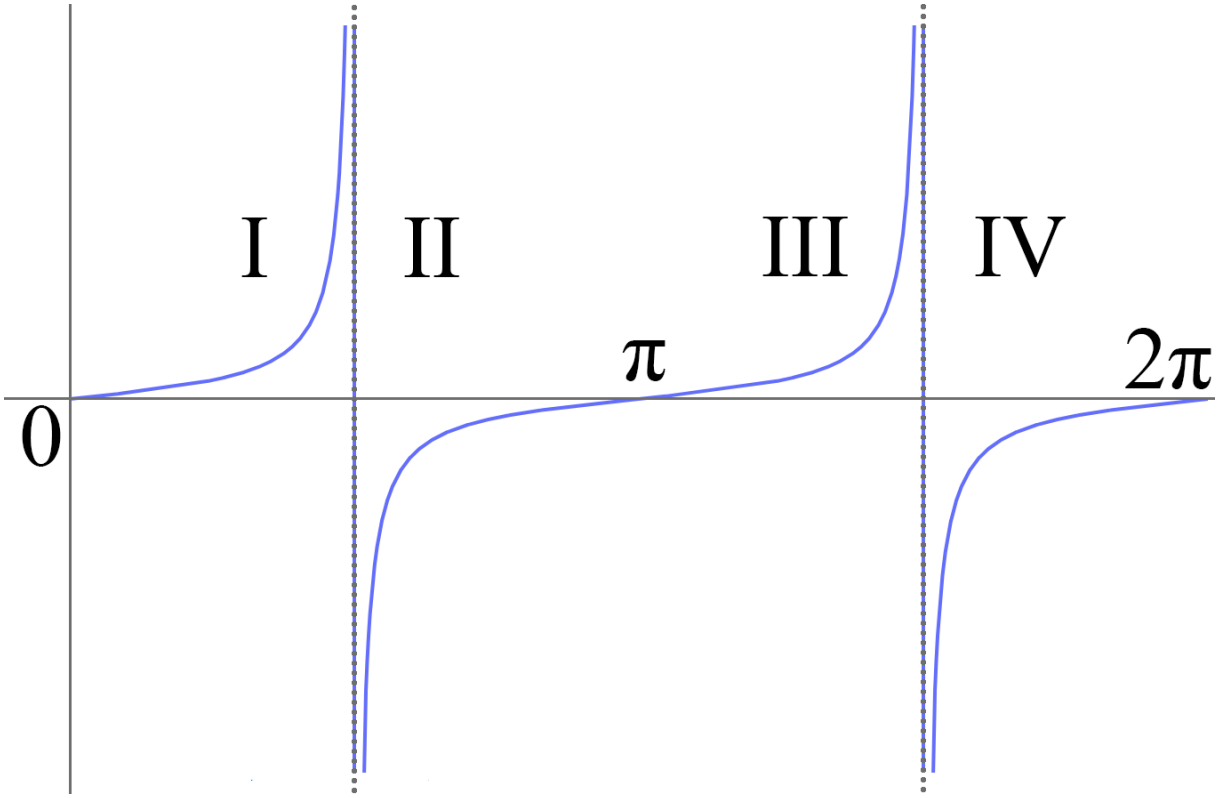

Figura 4.2 – Simetrias da função tangente.

maneira direta leva à necessidade de um número muito grande de termos, especialmente valores próximos aos pontos de descontinuidade (KANTABUTRA, 1996).

Antes de aplicar a interpolação, os ângulos são normalizados para o intervalo $[0, 2\pi]$ e convertidos para a região $[0, \pi/2]$ conforme o quadrante correspondente, garantindo a monotonicidade da função e o correto uso da tabela (STEWART J.D., 2021). Além disso, de maneira similar ao procedimento realizado com as funções seno e cosseno, certas equivalências da função tangente podem ser utilizadas para reduzir o intervalo de tabelamento, como mostrado na Figura 4.2. A interpolação linear é então utilizada para estimar valores intermediários. Sendo assim, as aproximações não funcionam de maneira adequada para valores próximos à $\pm\pi/2$ e $\pm 3\pi/2$.

4.5 ARCO TANGENTE

A função arco tangente pode ser estimada ao registrar valores contidos em $[0, 1]$, já que é possível utilizar a identidade apresentada na Equação 34 para cobrir os valores que estão no intervalo $(1, \infty)$. Além disso, como a função arco tangente é ímpar (GELFAND I.M., 2001), é possível utilizar sua paridade para reduzir o intervalo de tabelamento, com a relação $f(-x) = -f(x)$ sendo utilizada entre as regiões I e II da figura 4.3. Logo, a partir do uso de sua paridade, é possível lidar também com valores no intervalo $(-\infty, -1)$.

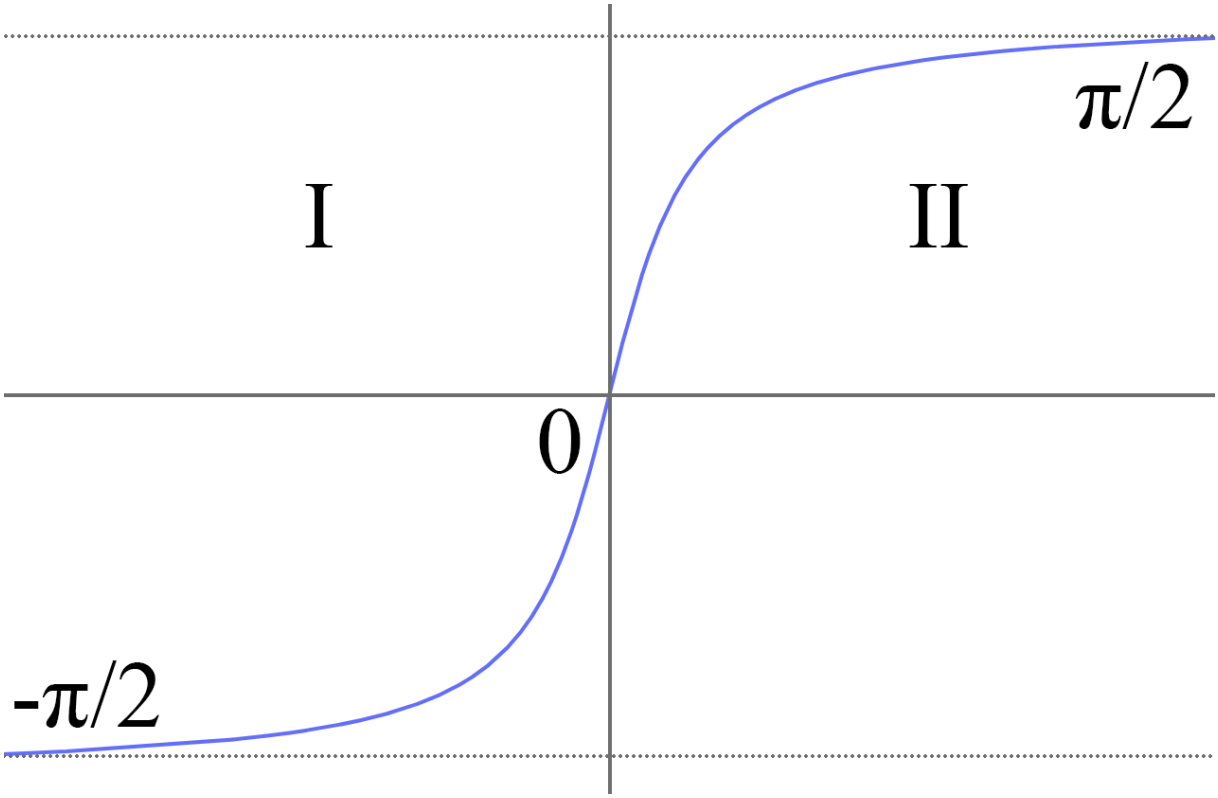

Figura 4.3 – Simetrias da função arco tangente.

$$\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right), \quad \text{para } x > 1 \tag{34}$$

Isso permite que a tabela de valores seja construída apenas para o intervalo $x \in [0, 1]$, com os demais valores sendo tratados por simetria e identidade. Como a função é ímpar, também se aplicou $f(-x) = -f(x)$, reduzindo ainda mais o intervalo necessário.

4.6 EXPONENCIAL

A função exponencial não pode ser tabelada de maneira similar às anteriores, uma vez que não é cíclica, fazendo com que sua aproximação através do método do tabelamento só possa ser utilizada em intervalos pré-determinados. Devido ao crescimento exponencial da função, qualquer tentativa de tabelamento exigiria um intervalo artificialmente limitado, resultando em perda significativa de precisão ou uso excessivo de memória (SEBAH P., 2002). Sendo assim, no contexto apresentado nesse trabalho, recomenda-se que a aproximação da função exponencial seja realizada utilizando um dos dois outros métodos apresentados nesse trabalho.

4.7 TANGENTE HIPERBÓLICA

A função tangente hiperbólica pode ser tabulada no intervalo $x \in [0, 2]$, aproveitando sua simetria ímpar para reduzir o esforço computacional. Valores acima de 2 podem ser aproximados pro 1, enquanto valores abaixo de -2, por -1.

4.8 SIGMOIDE

Com uma definição muito similar à tangente hiperbólica, a função sigmoide também apresenta saturação nos extremos, tendendo a 1 para entradas positivas de valor elevado e a 0 para entradas negativas elevadas em módulo (KYURKCHIEV N., 2015). Por isso, é possível utilizar uma tabela que contém o intervalo $x \in [0, 6]$, sendo os valores negativos reconstruídos por meio da identidade $f(-x) = 1 - f(x)$ (KYURKCHIEV N., 2015).

5 ALGORITMO CORDIC

Criado em 1959 por J. E. Volder, o *COordinate Rotation DIgital Computer*, ou *CORDIC* é um método de aproximação de funções trigonométricas, hiperbólicas e lineares através de um processo iterativo (AGUIAR R.G., 2020).

5.1 MODO CIRCULAR

Focando inicialmente no cálculo de funções trigonométricas com o método *CORDIC*, é possível determinar qual será o ponto resultante de uma rotação de *phi*. Isto é, partindo de um ponto dado por (x_0, y_0), rodando ϕ radianos no sentido horário, é possível descobrir qual é o ponto resultante da rotação, dado por (x_1, y_1), através de uma matriz de transformação (PELLEGRINI, 2015). Sendo assim, temos que:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = R \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{35}$$

A matriz de rotação R para uma volta no sentido horário, dada em 35, é dada por (PELLEGRINI, 2015):

$$R = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{36}$$

Para que a matriz de rotação seja útil para o uso no método CORDIC, é necessário que certas modificações sejam feitas (PARRIS, 2010). Primeiramente, ao fatorar R por $\cos(\phi)$:

$$R = \cos(\phi) \begin{bmatrix} 1 & \tan(\phi) \\ -\tan(\phi) & 1 \end{bmatrix} \tag{37}$$

Ainda é possível simplificar a matriz de rotação R ainda mais ao empregar somente ângulos de rotação ϕ tais que $tan(\phi_n) = 2^{-n}$. Em uma primeira análise, é possível pensar que essa operação aparentemente arbitrária limita os ângulos de rotação possíveis para se empregar no método *CORDIC*. No entanto, conforme n aumenta, os ângulos de rotação, dados por $arctan(2^{-n})$, caem de maneira rápida (ZHOU Z.L, 2023). Para o uso de um número N de iterações, é necessário guardar uma tabela de N elementos contendo os valores de $arctan(2^{-n})$.

Tendo em mente que o método *CORDIC* é feito em iterações, como o ângulo percorrido em cada passo cai de maneira acentuada, é possível se aproximar de todos os ângulos contidos em $[0, \pi/2]$ (ZHOU Z.L, 2023). Assim, é possível percorrer o círculo trigonométrico em sua totalidade ao se empregar certas simetrias sem a perda de precisão. Além disso, em implementações com hardware de recursos de processamento limitados, o cálculo de 2^{-n} é simples, sendo encontrado por um simples *bit-shift* (ZHOU Z.L, 2023). Como última modificação, agora temos que a matriz de rotação R é dada por:

$$R = \cos(\phi) \begin{bmatrix} 1 & 2^{-n} \\ -2^{-n} & 1 \end{bmatrix} \tag{38}$$

Como o método *CORDIC* é iterativo, a toda operação que se realiza, há a multiplicação por um fator $\cos(\phi)$, resultando em um valor cumulativo K_m dado por $K_m = \cos(\phi_1)\cos(\phi_2)\cos(\phi_3)\ldots\cos(\phi_m)$, com o índice m representando a última iteração. De acordo com (PARRIS, 2010), K_m, o produto desses fatores cossenoidais, converge de maneira rápida, sendo dado por:

$$\frac{1}{K_\infty} = \lim_{n\to\infty} \prod_{i=1}^{n} \sqrt{1 + 2^{-2i}} \approx 1.6467605 \tag{39}$$

O valor $\sqrt{1 + 2^{-2i}}$ representa a dilatação na magnitude provocada pela operação de rotação associada ao passo i, e seu acúmulo ao longo das iterações resulta no ganho global K_m. Sendo assim, é possível multiplicar o valor inicial para as iterações, nesse caso, o ponto $(1, 0)$, por um fator pré-calculado de valor $1/K_\infty$, transformando-o no ponto $(1/K_\infty, 0)$ (AGUIAR R.G., 2020). Em termos práticos, ao utilizar somente 10 termos, o erro absoluto entre $1/K_\infty$ e $1/K_{10}$ é de 0.00000050362, tornando a aproximação $1/K_m \approx 1/K_\infty$ válida para um número relativamente baixo de iterações. Assim, realizando a normalização no primeiro termo, a matriz R toma um formato mais simples, apresentado em 40.

$$R = \begin{bmatrix} 1 & 2^{-n} \\ -2^{-n} & 1 \end{bmatrix} \tag{40}$$

Ao voltar ao formato original, mostrado em 35, e ao generalizar os valores para um passo qualquer, temos:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = R \begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & 2^{-n} \\ -2^{-n} & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \tag{41}$$

E ao abrir o apresentado em 41:

$$x_{n+1} = x_n + 2^{-n} y_n \qquad \text{e} \qquad y_{n+1} = -2^{-n} x_n + y_n \tag{42}$$

Tendo em mente essa estrutura para uma rotação no sentido horário e, a partir de manipulações matriciais e algébricas extremamente similares para o caso de uma rotação no sentido anti-horário, temos que a matriz de rotação é dada por (ZHOU Z.L, 2023):

$$R_{anti-horário} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{43}$$

Chegando em equações para a relação entre termos subsequentes de uma rotação no sentido anti-horário, temos (PELLEGRINI, 2015):

$$x_{n+1} = x_n + -2^{-n} y_n \qquad \text{e} \qquad y_{n+1} = 2^{-n} x_n + y_n \tag{44}$$

Sendo assim, ao fim das iterações, é possível encontrar os valores aproximados das funções seno e cosseno através de:

$$x_{final} \approx \cos(\phi) \qquad \text{e} \qquad y_{final} \approx \sin(\phi) \tag{45}$$

Por fim, há de se considerar o papel de z_n no método apresentado. Conforme os termos x_{n+1} e y_{n+1} mostrados em 42 e 44 são alterados nas iterações do método, z_{n+1} também deve ser alterado (PARRIS, 2010). O mesmo representa a diferença residual entre o ângulo da iteração atual e o ângulo desejado, sendo inicializado com o ângulo buscado. Ou seja, se a conta buscada é $\sin(\phi)$ ou $\cos(\phi)$, inicia-se com $z_0 = \phi$. Quanto mais próximo z_n está de zero, mais próximo o ângulo encontrado será do real (ZHOU Z.L, 2023). Seu formato é dado em 46.

$$z_{n+1} = z_n \pm \arctan(2^{-n}) \tag{46}$$

Sendo que o termo $\arctan(2^{-n})$ em 46 é negativo em z_{n+1} caso $z_n > 0$ e positivo se $z_n < 0$. Sendo assim, é possível relacionar as expressões para x_{n+1}, y_{n+1} e z_{n+1} de modo que:

$$\begin{cases} x_{n+1} = x_n - d \cdot 2^{-n} \cdot y_n \\ y_{n+1} = y_n + d \cdot 2^{-n} \cdot x_n \\ z_{n+1} = z_n - d \cdot \arctan(2^{-n}) \end{cases} \tag{47}$$

Sendo o termo $d = 1$ na Equação 47 caso $z_n > 0$ e se $d = -1$ caso caso $z_n < 0$.

5.2 MODO VETORIAL

O cálculo de $\arctan(m)$ por CORDIC baseia-se no modo de *vectoring* ou modo vetorial, adaptado para determinar o ângulo cujo tangente seja igual à magnitude de entrada (ZHOU Z.L, 2023). A ideia consiste em iniciar com o vetor $(x_0, y_0) = (1,\ m)$ e iterativamente rotacioná-lo em pequenos incrementos angulares de forma a reduzir progressivamente a componente y a zero, acumulando em z a soma dos ângulos correspondentes. Ao final, z aproxima $\arctan(m)$ para $|m| \leq 1$. Essa técnica segue a mesma lógica de iterações do modo circular para seno e cosseno, mas invertendo o papel de reduzir y e acumular o ângulo em z para o cálculo de arctan diretamente (PARRIS, 2010; ZHOU Z.L, 2023). Para garantir boa convergência quando m pode exceder 1, aplica-se primeiro uma normalização baseada na identidade em 48 (GELFAND I.M., 2001).

$$\arctan(m) = \begin{cases} \dfrac{\pi}{2} - \arctan\Big(1/m\Big), & m > 1, \\ -\dfrac{\pi}{2} - \arctan\Big(1/m\Big), & m < -1, \\ \arctan(m), & |m| \leq 1. \end{cases} \tag{48}$$

Dessa forma, quando $|m| > 1$, calcula-se primeiramente $\arctan(1/|m|)$ com $|1/m| < 1$ e ajusta-se o resultado via $\pm\pi/2$ conforme o sinal de m. Essa normalização assegura que a iteração principal ocorra sempre com magnitude de entrada $M = |m| \leq 1$, evitando convergência lenta ou necessidade de muitas iterações (ZHOU Z.L, 2023). Após a normalização, assume-se $M = |m| \leq 1$ e define-se as variáveis iniciais.

$$x_0 = 1, \quad y_0 = M, \quad z_0 = 0. \tag{49}$$

Em cada iteração $n = 0, 1, \ldots, N-1$, escolhe-se o sinal dos termos em uma iteração qualquer n, denotado por d_n, a partir de 50, de modo a reduzir a magnitude de y_n.

$$d_n = \begin{cases} +1, & y_n \geq 0, \\ -1, & y_n < 0, \end{cases} \tag{50}$$

Então realiza-se a rotação incremental pelo ângulo $\arctan(2^{-n})$, sendo o procedimento geral utilizado dado em 51 (ZHOU Z.L, 2023):

$$\begin{cases} x_{n+1} = x_n + d_n\, 2^{-n}\, y_n, \\ y_{n+1} = y_n - d_n\, 2^{-n}\, x_n, \\ z_{n+1} = z_n + d_n\, \arctan\left(2^{-n}\right) \end{cases} \tag{51}$$

O termo 2^{-n} é obtido via tabela pré-computada, assim como a tabela de $\arctan(2^{-n})$ para $n = 0, \ldots, N-1$. Ao término de N iterações, y_N estará próximo de zero e z_N aproximará $\arctan(M)$ dentro do erro de truncamento de iterações e da precisão de armazenamento dos valores relevantes (AGUIAR R.G., 2020). Para $m < 0$, aplica-se o sinal final: $\arctan(m) = -\arctan(|m|)$ após obter $\arctan(|m|)$ conforme o processo descrito acima (PARRIS, 2010). É importante observar que, no modo de cálculo de arctan via CORDIC, ou seja, o modo vetorial, não se aplica fator de correção de ganho (K), pois o procedimento visa reduzir y_n a zero, acumulando diretamente o ângulo em z_n, de maneira que a normalização adicional da magnitude do vetor não seja necessária (ZHOU Z.L, 2023).

5.3 MODO HIPERBÓLICO

O modo hiperbólico do CORDIC permite calcular funções como $\sinh(\theta)$, $\cosh(\theta)$, $\tanh(\theta)$, exponenciais e logaritmos por meio de um processo iterativo, análogo ao modo circular, mas trabalhando em coordenadas hiperbólicas (ZHOU Z.L, 2023). A matriz de transformação que relaciona uma rotação de θ radianos entre dois pontos no plano hiperbólico é dada em 52 (PELLEGRINI, 2015).

$$H = \begin{bmatrix} \cosh(\theta) & \sinh(\theta) \\ \sinh(\theta) & \cosh(\theta) \end{bmatrix} \tag{52}$$

De maneira similar ao procedimento feito em 37, é possível simplificar a matriz H ao fatorar $\cosh(\theta)$ (PARRIS, 2010).

$$H = \cosh(\theta) \begin{bmatrix} 1 & \tanh(\theta) \\ \tanh(\theta) & 1 \end{bmatrix} \tag{53}$$

Escolhendo ângulos de rotação θ_i tais que $\tanh(\theta_i) = 2^{-i}$, ou seja, $\theta_i = \operatorname{arctanh}(2^{-i})$, podemos simplificar a matriz de rotação hiperbólica ainda mais. Assim como no caso do modo circular, os valores de $\tanh(\theta_i)$ decaem rapidamente conforme i cresce. Esses valores $\operatorname{artanh}(2^{-i})$ são pré-calculados e armazenados em uma tabela de N elementos, isto é, salvo uma exceção que será explorada nos próximos parágrafos, um valor é armazenado por iteração, permitindo aproximar θ pela soma ou subtração iterativa de θ_i por meio de pequenas rotações hiperbólicas (MOROZ L., 2018). A formulação matricial de cada passo é dada em 54.

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & d_n\, 2^{-n} & 0 \\ d_n\, 2^{-n} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -d_n \operatorname{arctanh}(2^{-n}) \end{bmatrix} \tag{54}$$

Em 54, z_n é o ângulo residual hiperbólico, inicializado em $z_0 = \theta$. $d_n = \pm 1$ é escolhido para reduzir $|z_n|$ a cada iteração, subtraindo ou adicionando $\operatorname{artanh}(2^{-n})$ de z_n (ZHOU Z.L, 2023). Sendo assim, a lógica de implementação em código do processo iterativo descrito é dada a partir de 55.

$$\begin{cases} x_{n+1} = x_n + d_n\, 2^{-n}\, y_n, \\ y_{n+1} = y_n + d_n\, 2^{-n}\, x_n, \\ z_{n+1} = z_n - d_n\, \operatorname{artanh}(2^{-n}) \end{cases} \qquad d_n = \begin{cases} +1, & z_n \geq 0, \\ -1, & z_n < 0. \end{cases} \tag{55}$$

Agora, de maneira distinta do método de cálculo explicado em 5.1, não é necessário aplicar um fator de compensação do erro para o cálculo das funções tangente hiperbólica, sigmoide e exponencial real (MOROZ L., 2018), já que ambas são calculadas a partir da razão entre x e y, que acumulam um erro idêntico (ERCEGOVAC M.D., 2003). É relevante notar que diferentemente do método de cálculo explicado em 5.1 a forma de se compensar o ganho incremental em cada iteração n no modo hiperbólico, é um pouco mais indireta. De maneira similar ao caso do modo circular, cada rotação no modo hiperbólico multiplica o vetor x_n, y_n por um fator $\sqrt{1 - 2^{-2i}}$ (ERCEGOVAC M.D., 2003). No entanto, se aplicarmos diretamente a mesma lógica do fator K do modo circular, como mostrado em 56, obtemos um valor incorreto para o ganho acumulado no modo hiperbólico.

$$\frac{1}{J_\infty} \neq \lim_{n\to\infty} \prod_{i=1}^{n} \sqrt{1 - 2^{-2i}} \tag{56}$$

Isso ocorre porque, para certos ângulos, o valor de $\text{arctanh}(2^{-i})$ é menor que seus sucessores. Ou seja, para certos valores do índice i, uma rotação pode ser "cancelada"pelas subsequentes, fazendo com que o algoritmo não converja (ERCEGOVAC M.D., 2003). Uma forma de sanar esse problema é a repetição dos índices sujeitos a esse problema, pode-se adotar um critério para a realização de uma rotação somente se o valor absoluto do ângulo residual z_i for maior que $\text{arctanh}(2^{-i})$. Dessa forma, rotações com impacto desprezível no resultado final são descartadas, otimizando a execução e simplificando o controle do laço iterativo. Os índices que podem ocasionar este problema são dados por 57 e podem ser salvos em código.

$$\text{arctanh}(2^{-i}) < \sum_{k=i+1}^{\infty} \text{arctanh}(2^{-k}) \tag{57}$$

Alguns valores que satisfazem as condições acima são $\{4, 13, 40, 121, 364, 1093, ...\}$ (ERCEGOVAC M.D., 2003). Outra forma de tratar os índices cujas rotações são menores que as suas sucessoras, que foi o método empregado no código que será mostrado no próximo capítulo, é pular as rotações tais que $|z_i| < \text{arctanh}(2^{-i})$. Essa condição também contribui para a convergência sem precisar listar explicitamente repetições de índices, pois passamos apenas por índices onde a rotação resulta em uma diferença notável (ERCEGOVAC M.D., 2003).

O processo iterativo básico do modo hiperbólico, indicado em (55), converge de forma estável para o cálculo de $\tanh(z)$ quando $|z|$ está dentro de um intervalo limitado, como $|z| \leq 1$) (ERCEGOVAC M.D., 2003). Para $|z| > 1$, aplicar diretamente o CORDIC hiperbólico pode exigir muitas iterações ou mesmo sair da região de convergência ideal. Emprega-se então a redução de intervalo via identidade de duplicação de argumento, levando o argumento para faixa onde o algoritmo converge bem. Em seguida, o valor original é reconstruído (ERCEGOVAC M.D., 2003). A identidade de duplicação para tangente hiperbólica é dada em 58 (CANNON J.W., 1997).

$$\tanh(z) = \frac{2\,\tanh(z/2)}{1 + \tanh^2(z/2)}. \tag{58}$$

Dessa forma, se $|z| > 1$, considera-se $z_1 = z/2$ e obtém-se $\tanh(z)$ a partir de $\tanh(z_1)$. Caso $|z_1| > 1$, repete-se a redução de intervalo até obter z_k tal que $|z_k| \leq 1$. Em seguida, aplica-se o processo iterativo do modo hiperbólico base para calcular

$\tanh(z_k)$. Finalmente, reconstrói-se $\tanh(z)$ via iterações reversas da identidade (58). O número de reduções necessárias é aproximadamente $\lceil \log_2(|z|) \rceil$, assegurando $|z/2^k| \leq 1$ (ERCEGOVAC M.D., 2003).

Para calcular e^z, usa-se a relação entre tanh e exponenciais (CANNON J.W., 1997). Partindo da definição de $\tanh(x)$, tem-se em 59.

$$\tanh\Big(r/2\Big) = \frac{e^{r/2} - e^{-r/2}}{e^{r/2} + e^{-r/2}} = \frac{e^r - 1}{e^r + 1}, \tag{59}$$

E da Equação 59 segue a Equação 60.

$$e^r = \frac{1 + \tanh(r/2)}{1 - \tanh(r/2)}, \quad \text{desde que } |\tanh(r/2)| < 1. \tag{60}$$

Aplica-se então redução de intervalo em z por múltiplos de $\ln 2$, mostrada em 61 (ERCEGOVAC M.D., 2003).

$$z = n\,\ln 2 + r, \quad n = \left\lfloor \frac{z}{\ln 2} + 0.5 \right\rfloor, \quad r = z - n\,\ln 2, \tag{61}$$

Após o arredondamento de n, realiza-se uma correção adicional em r, caso $|r| > \ln 2/2$, para manter o argumento de $\tanh(r/2)$ suficientemente pequeno. Essa operação melhora a estabilidade e precisão numérica na região próxima à descontinuidade da tangente hiperbólica, e é realizada ajustando n em ± 1, conforme necessário. Em seguida, calcula-se $\tanh(r/2)$ pelo CORDIC hiperbólico base, com $|r/2| \leq 0.25\,\ln 2 < 1$), obtendo-se **??**.

$$e^r = \frac{1 + \tanh(r/2)}{1 - \tanh(r/2)}. \tag{62}$$

Finalmente, reconstrói-se $e^z = 2^n\,e^r$, realizando multiplicações ou divisões por 2 de forma exata. Essa abordagem conecta diretamente o cálculo de tanh pelo CORDIC hiperbólico ao cálculo de exponencial, através do uso da tabela de valores de $\operatorname{artanh}(2^{-i})$ (MOROZ L., 2018).

Partindo para a função sigmoide logística, temos que sua definição é dada por 63 (BURKOV, 2019).

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{63}$$

Para implementá-la via CORDIC, calcula-se e^{-z} conforme descrito acima e obtém-se $\sigma(z)$ diretamente a partir de $\frac{1}{1+e^{-z}}$. Além disso, para valores de entrada z com módulo superior a 10, a implementação considera que a função $\sigma(z)$ já se encontra suficientemente próxima de seus limites assintóticos 0 ou 1. Assim, o valor de saída é fixado como 0 para $z < -10$ e 1 para $z > 10$, evitando o uso desnecessário de recursos computacionais. Essa heurística de saturação é bastante comum em sistemas embarcados, onde é necessário conciliar precisão e desempenho.

6 IMPLEMENTAÇÃO EM CÓDIGO E COMPARAÇÃO DOS RESULTADOS

A fim de analisar os resultados obtidos utilizando os métodos descritos anteriormente, foram feitas as implementações no processador SAPHO dos métodos descritos acima. Para funções periódicas, como seno, cosseno e tangente, foram utilizados valores de entrada em radianos no intervalo $[0, 6.3]$. O valor 6.3 foi escolhido pois as aproximações englobam o valor 2π, tendo em mente que $2\pi \approx 6,283$. Os erros e tempos de convergência em número de ciclos de *clock* serão mostrados posteriormente.

Já para as funções não periódicas, como arco tangente, tangente hiperbólica e sigmoide, foram utilizadas entradas pertencentes ao intervalo $[-2, 2]$, com passos de 0.01. Assim como no caso anterior, a precisão e o número de ciclos de *clock* necessários para a convergência foram analisados.

Os testes foram feitos utilizando dois conjuntos distintos de parâmetros para os processadores. O primeiro, representado pelo traçado laranja nos gráficos de erro utiliza 32 bits na unidade lógica aritmética (ULA) do processador, com 8 bits para a representação do expoente, 23 bits para a representação da mantissa e 1 bit para o sinal. Enquanto nas figuras azuis, foram utilizados processadores com 23 bits na ULA, com 6 bits para a representação do expoente, 16 bits para a representação da mantissa e 1 bit para o sinal.

É importante ressaltar que utilizou-se a configuração de 23 bits na ULA em um conjunto de testes porque uma mantissa de 16 bits oferece uma incerteza de $u \approx \frac{1}{2}2^{-16}$, ou cerca de 0.000008, suficiente para certas aplicações das funções avaliadas. Do ponto de vista de implementação, a ULA de 23 bits vê uma diminuição considerável no uso de recursos lógicos, enquanto diminui a precisão das estimativas (SANTOS V.A.M., 2019).

Inicialmente, as funções foram implementadas na linguagem C, onde a validade dos códigos foi testada. Posteriormente, as funções foram modificadas para que se adequassem à sublinguagem de C utilizada pelo processador. Então, os resultados foram comparados com o valor real de cada função em cada ponto testado. Também foi calculado o número médio de ciclos do processador utilizado para a realização das contas.

Como explicitado no capítulo 2, o processador *SAPHO* é um processador *softcore*, que tem seus parâmetros modificáveis (VICCINI, 2023). Todos os testes foram realizados com os seguintes parâmetros:

- Tamanho da Pilha de Dados = 5

- Tamanho da Pilha de Subrotinas = 5
- Número de Portas de Entrada = 1
- Número de Portas de Saída = 1
- Nível de Pipeline do Processador = 3
- Valor de normalização de ponto fixo = 128

6.1 MACLAURIN

Como explicado no Capítulo 3, as aproximações através de séries de Maclaurin são montadas a partir da expressão que relaciona dois termos subsequentes. Após passos de normalização do valor de entrada, quando necessário, começa o processo iterativo de estimação do valor da função em um determinado ponto, até que o próximo termo calculado tenha valor menor do que a tolerância definida em código, de valor 0.000001 para todos os casos apresentados. As funções periódicas foram testadas no intervalo $[0, 6.3]$, enquanto as funções não periódicas foram testadas no intervalo $[-2, 2]$.

6.1.1 Seno, Cosseno e Tangente

O Código 6.1, foi utilizado para exemplificar o processo de cálculo da função através de Séries de Maclaurin. Ele chama a função *normalizador de ângulo*, que deixa a entrada no intervalo $[-\pi, \pi]$. Então, calcula-se o valor de cada termo da função de forma iterativa, como mostrado na Equação 6. Então, o valor de cada termo calculado é somado em uma variável de resultado. Quando o valor de um termo é menor que o valor de tolerância definido previamente, o processo é interrompido e a função retorna o valor calculado.

Utilizando o Código 6.1, foi obtido o erro mostrado na Figura 6.1. Nota-se que o erro é relativamente uniforme ao longo do intervalo testado, especialmente para a aproximação com 32 bits.

Código 6.1 – Função Seno.

```
1 float seno(float x) {
2     x = normalizador_de_angulo(x);
3     float termo = x;
4     float resultado = termo;
5     float tolerancia = 0.000001;
6     int j = 1;
7     while (absoluto(termo) > tolerancia) {
8         termo = termo * (- x * x / ((j + 1) * (j + 2)));
```

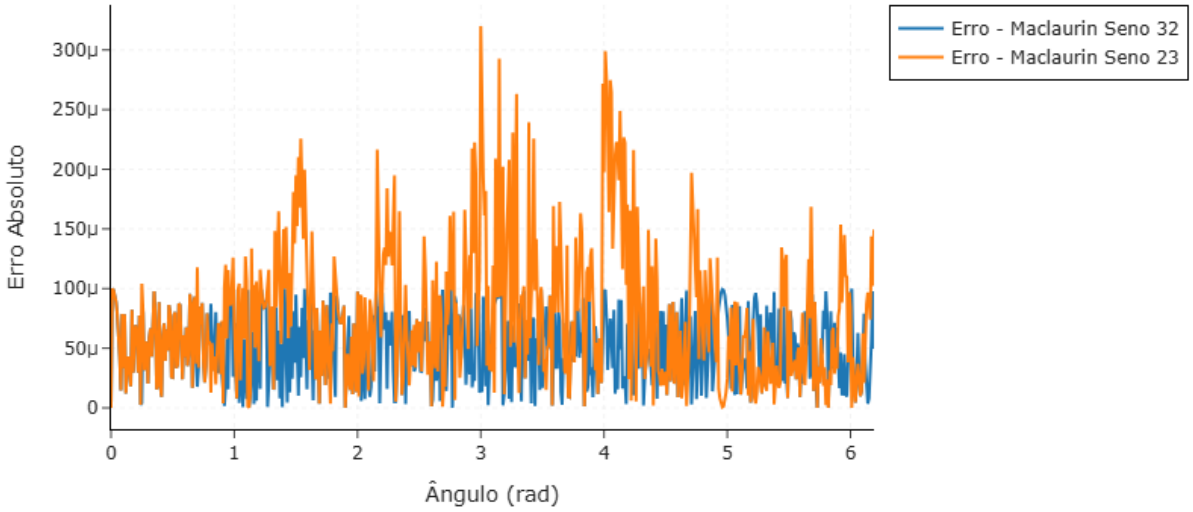

Figura 6.1 – Erro da função seno através de séries de Maclaurin.

```
9           resultado = resultado + termo;
10          j = j + 2;
11      }
12      return resultado;
13  }
```

Já o gráfico que apresenta o erro nas aproximações de Maclaurin de cosseno está disponível na Figura 6.2. Seu erro se comporta de maneira extremamente similar à função seno, mas apresentando um erro médio maior, já que as aproximações de Maclaurin que encontram resultados menores tendem à zero mais rapidamente na média (BOYCE W.E., 2017) .

Já a função tangente tem seus resultados mostrados na Figura 6.3. Como a função tem descontinuidades em valores $x = (2n + 1)(\pi/2)$, com $n = 1, 2, 3, 4 \cdots$, foi mostrado o erro normalizado para valores maiores do que 100. O erro no intervalo de $[0, 1.2]$ foi mostrado na Figura 6.4.

6.1.2 Arco Tangente, Exponencial, Tangente Hiperbólica e Sigmoide

Já as funções arco tangente, exponencial, tangente hiperbólica e sigmoide, que não são periódicas, foram testadas no intervalo $[-2, 2]$, com um passo de 0.01. Começando com a função arco tangente, é possível ver o seu resultado na Figura 6.5.

Seu erro é relativamente uniforme no intervalo testado quando se utiliza 32 bits. No entanto, ao analisar os valores próximos de ± 0.5 e ± 2 para a aproximação de 23 bits,

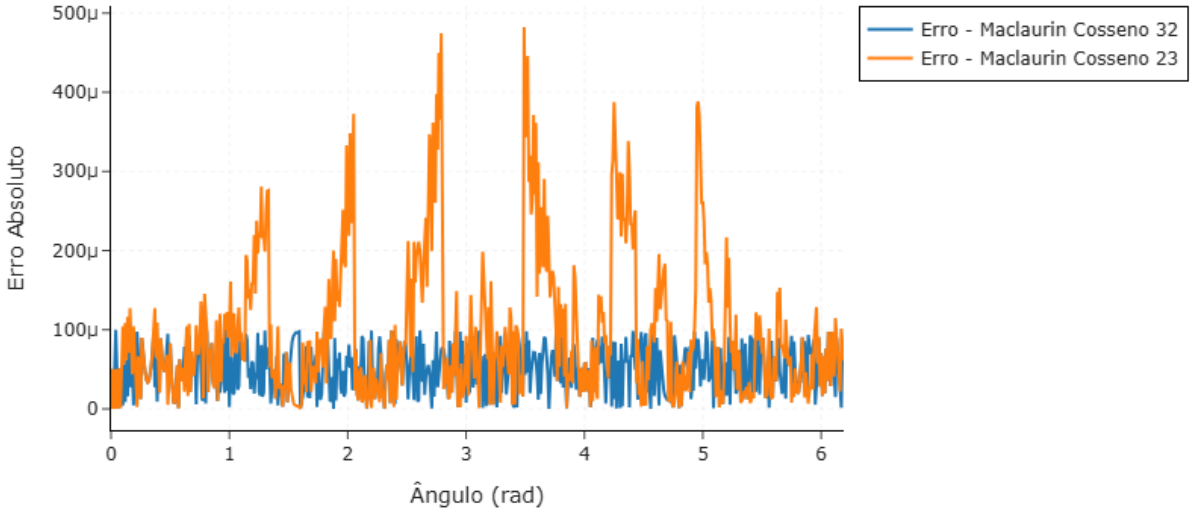

Figura 6.2 – Erro da função cosseno através de séries de Maclaurin.

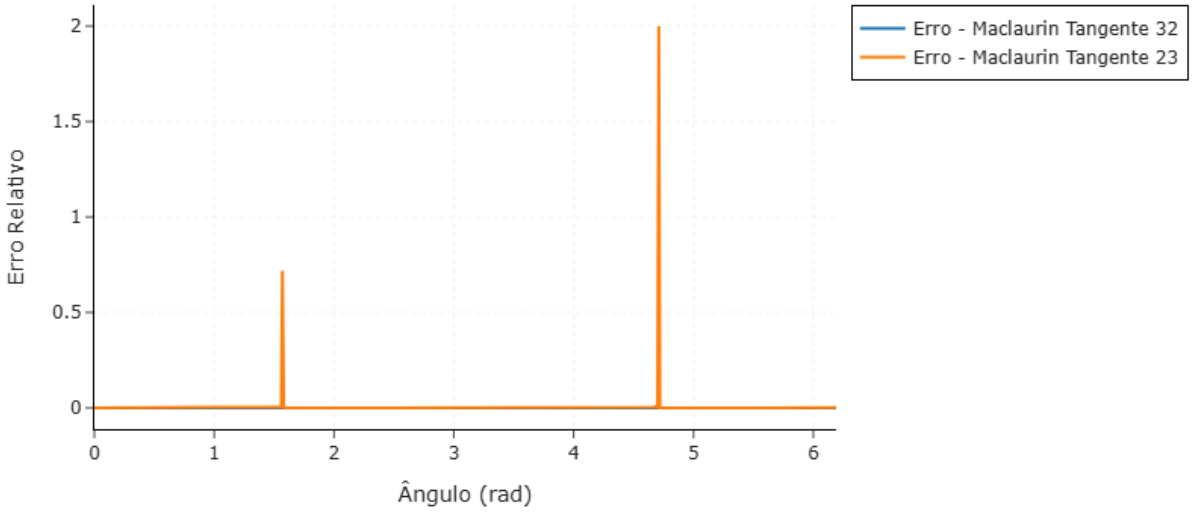

Figura 6.3 – Erro da função tangente através de séries de Maclaurin.

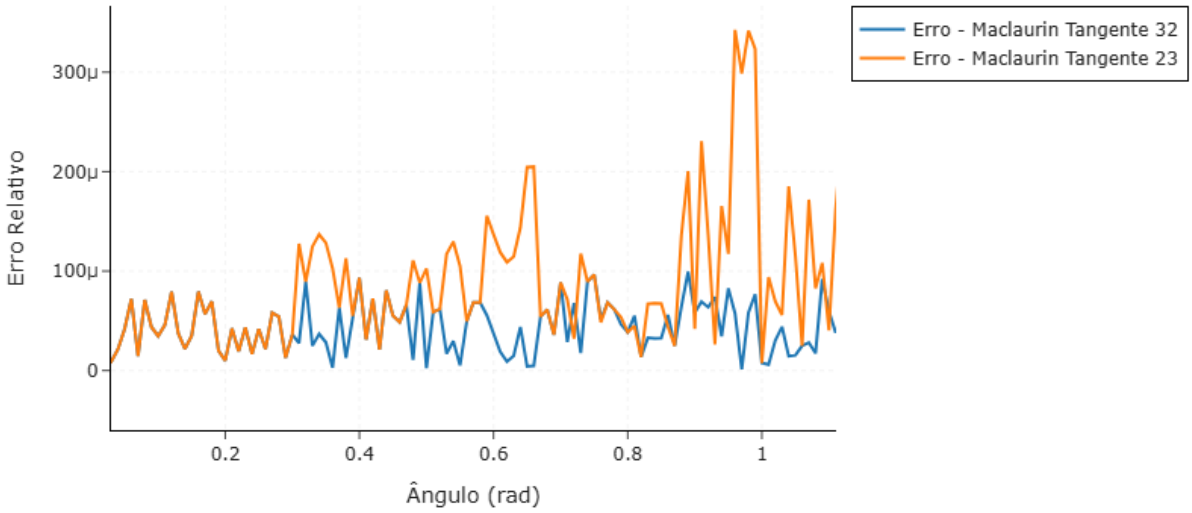

Figura 6.4 – Erro da função tangente através de séries de Maclaurin Ampliada.

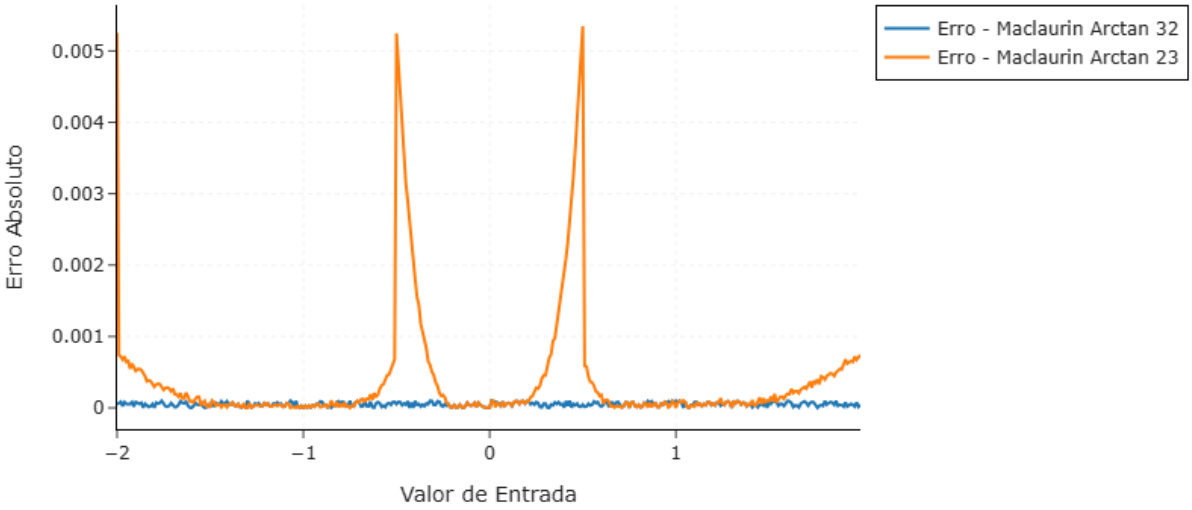

Figura 6.5 – Erro da função arco tangente através de séries de Maclaurin.

Figura 6.6 – Erro da função exponencial através de séries de Maclaurin.

nota-se um aumento no erro absoluto. Isso ocorre pois há a aplicação de uma identidade trigonométrica para acelerar a convergência da função. Sem a aplicação dessa identidade, são necessários mais termos para a convergência da série. Mesmo assim, para valores próximos de ± 0.5 e ± 2, mais somas em ponto flutuante são feitas, aumentando o erro da aproximação.

Para a função exponencial, podemos ver o comportamento de seu erro na Figura 6.6. É possível notar que seu erro absoluto é baixo, mas aumenta conforme o valor da função cresce para a representação de 23 bits. Esse comportamento é explicado pela tolerância utilizada. Tem-se que, quando um termo tem valor absoluto menor do que a tolerância aplicada, a aproximação é interrompida e seu resultado final é determinado. Mas, como na representação de 23 bits a tolerância deve ser mais alta, para determinados intervalos de aproximadamente 0.307, a função retorna o mesmo valor, já que os termos subsequentes da série são menores que a tolerância. Esse fator ocorre porque a série de Maclaurin da função exponencial converge de maneira bem rápida, com seus termos diminuindo em valor absoluto de maneira acelerada (BOYCE W.E., 2017).

Enquanto para a função tangente hiperbólica, é possível analisar seu erro mostrado na Figura 6.7. Como apresentado em 27, sua aproximação depende da aproximação da função exponencial. Sendo assim, seu erro está diretamente relacionado com o erro da função exponencial. Logo, assim como na aproximação da função exponencial, é possível ver que o erro é similar à função exponencial, crescendo conforme o valor de entrada cresce para a representação em 23 bits, enquanto a representação de 32 bits permanece estável.

Por fim, analisando a função sigmoide, temos o gráfico 6.8. Assim como a função

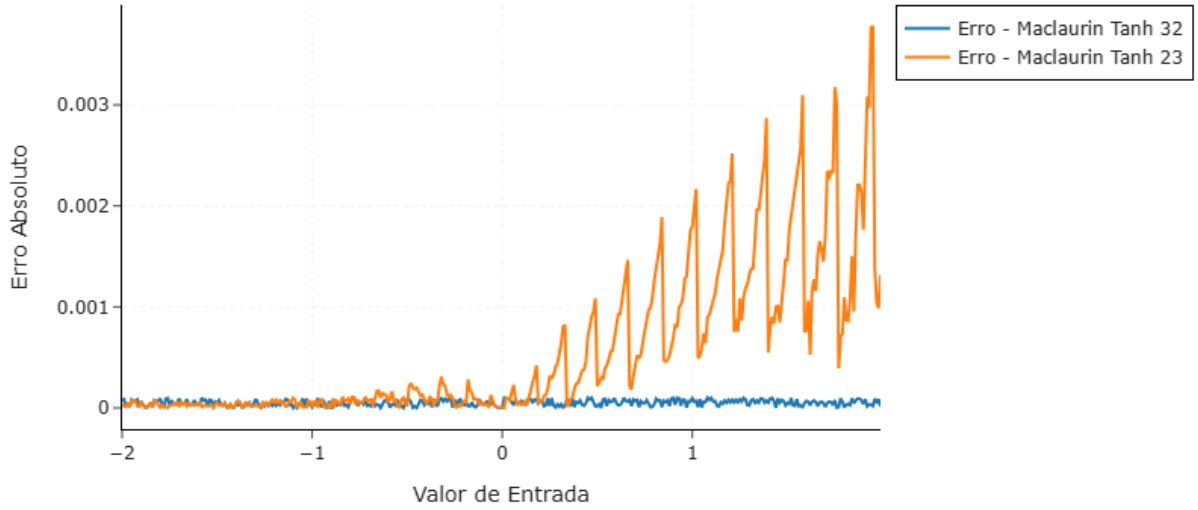

Figura 6.7 – Erro da função tangente hiperbólica através de séries de Maclaurin.

Figura 6.8 – Erro da função sigmoid através de séries de Maclaurin.

tangente hiperbólica, sua aproximação é relacionada à estimativa da função exponencial, portanto, quando o erro da função exponencial é maior, o seu erro também é maior.

6.2 TABELAMENTO

Como explicitado no Capítulo 4, foram criadas tabelas com os valores das funções de interesse, a fim de comparar os resultados obtidos. Com exceção da função exponencial, que não é periódica nem limitada (BOYCE W.E., 2017), levando à impossibilidade de aproximá-la através de tabelas excluindo o caso de sua utilização em um intervalo pré-determinado, as funções periódicas foram testadas no intervalo $[0, 6.3]$ e as funções não

periódicas foram testadas no intervalo $[-2, 2]$, para que seja possível comparar os resultados obtidos com tabelamento entre aproximações por Maclaurin e tabelamento. Para todas as funções, foram utilizadas tabelas com 48 valores das funções relevantes, já que, com 48 valores é possível, teoricamente, obter um erro médio de 0.00005 para as funções seno e cosseno, já que, a partir desse valor, aumentos no número de itens na tabela não alteram de maneira significativa a precisão da aproximação, uma vez que os erros introduzidos pela representação de ponto flutuante começam a dominar o erro na representação. A Tabela 6.1 mostra os intervalos utilizados no tabelamento e quais técnicas foram empregadas para a redução do intervalo armazenado.

Tabela 6.1 – Resumo dos intervalos de tabelamento utilizados.

Função	Intervalo Tabelado	Estratégias Adicionais
Seno	$[0, \pi/2]$	Simetria e periodicidade
Cosseno	$[0, \pi/2]$	Simetria e periodicidade
Tangente	—	Razão seno/cosseno
Arco Tangente	$[0, 1]$	Simetria, identidade recíproca
Exponencial	—	Não implementado
Tangente Hiperbólica	$[0, 2]$	Saturação ± 1, simetria
Sigmoide	$[0, 6]$	Identidade $f(-x) = 1 - f(x)$

6.2.1 Seno, Cosseno e Tangente

O Código 6.2 servirá como exemplo da estrutura dos códigos de tabelamento utilizados. Analisando a função seno, encontra-se o erro mostrado na Figura 6.9. O seu erro médio apresenta uma distribuição um pouco mais dispersa do que da aproximação por séries de Maclaurin da função seno, mostrada na Figura 6.1.

Código 6.2 – Função Seno.

```
1  int LUT_Tamanho = 48;
2  float Seno_LUT[48] "Seno_LUT.txt";
3
4  float seno_LUT(float x) {
5      while (x < 0) x = x + 2.0 * pi;
6      while (x >= 2.0 * pi) x = x - 2.0 * pi;
7
8      int quadrante;
9      if (x <= pi/2.0) { quadrante = 1; }
10     else if (x <= pi) { quadrante = 2; x = pi - x; }
11     else if (x <= 3.0*pi/2.0) { quadrante = 3; x = x - pi; }
12     else { quadrante = 4; x = 2.0*pi - x; }
13
```

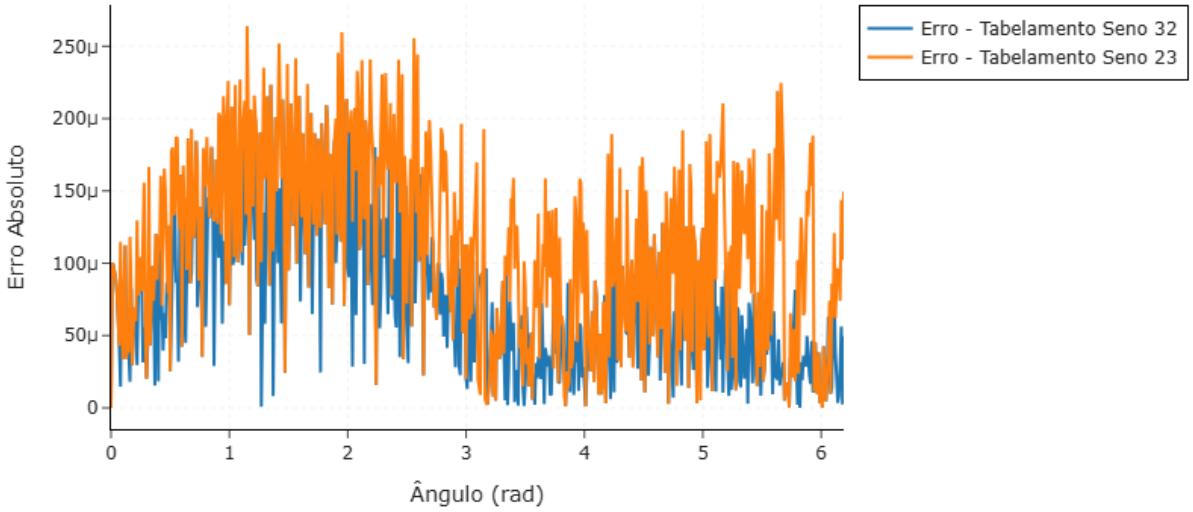

Figura 6.9 – Erro da função seno através de tabelamento.

```
14      float escala = (LUT_Tamanho - 1) / (pi/2.0);
15      float idxf = x * escala;
16      int idx = idxf;
17      float frac = idxf - idx;
18
19      if (idx > LUT_Tamanho - 2) { idx = LUT_Tamanho - 2; frac = 1.0; }
20
21      float v = Seno_LUT[idx] + (Seno_LUT[idx+1] - Seno_LUT[idx]) * frac;
22
23
24      if (quadrante == 1 || quadrante == 2) return v;
25      else return -v;
26  }
```

Já a função cosseno apresenta os resultados mostrados na Figura 6.10. Nota-se que seu erro tem comportamento bem similar à aproximação da função seno apresentada, porém defasada.

Por fim, para a função tangente, é possível obter seus valores a partir da razão das aproximações de seno e cosseno. Assim como no caso da aproximação por Maclaurin, o erro mostrado é o normalizado em torno das descontinuidades, já que a função assume valores extremamente altos quando próxima às suas descontinuidades (GELFAND I.M., 2001). Seu comportamento é mostrado na Figura 6.11. O seu comportamento fora do intervalo que apresenta a divergência é mostrado na Figura 6.12.

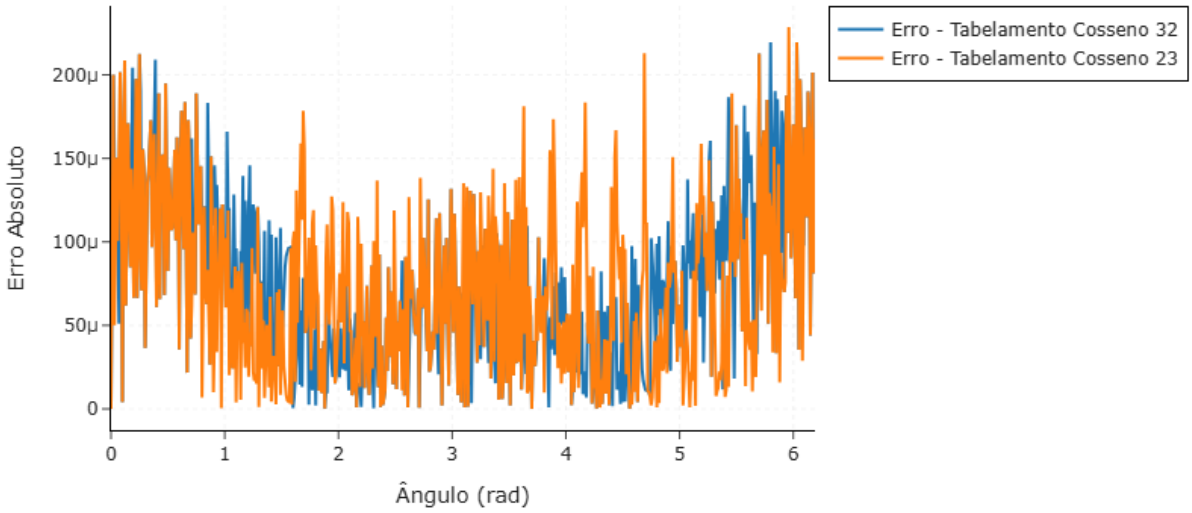

Figura 6.10 – Erro da função cosseno através de tabelamento.

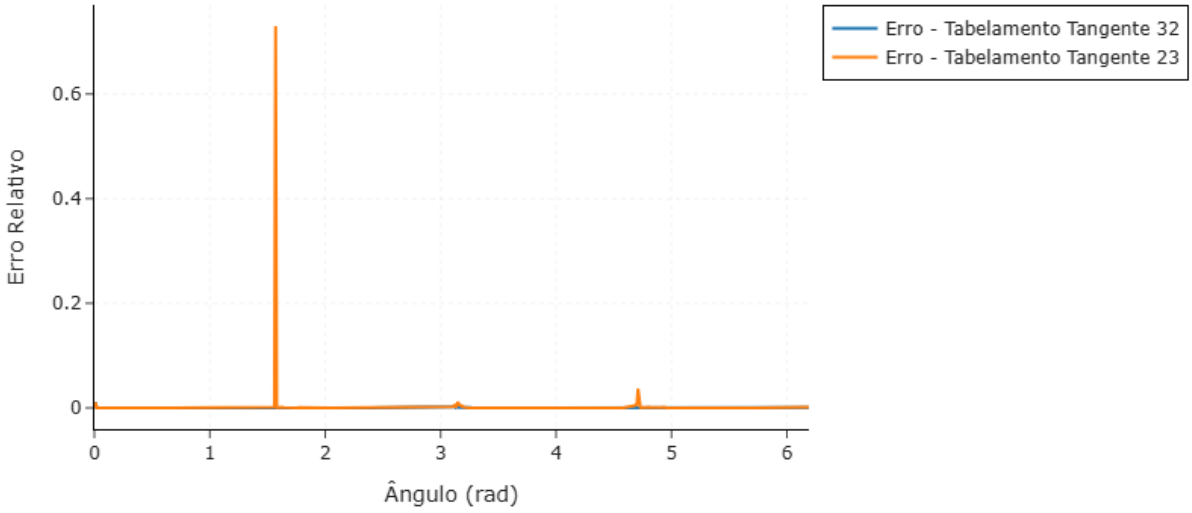

Figura 6.11 – Erro da função tangente através de tabelamento.

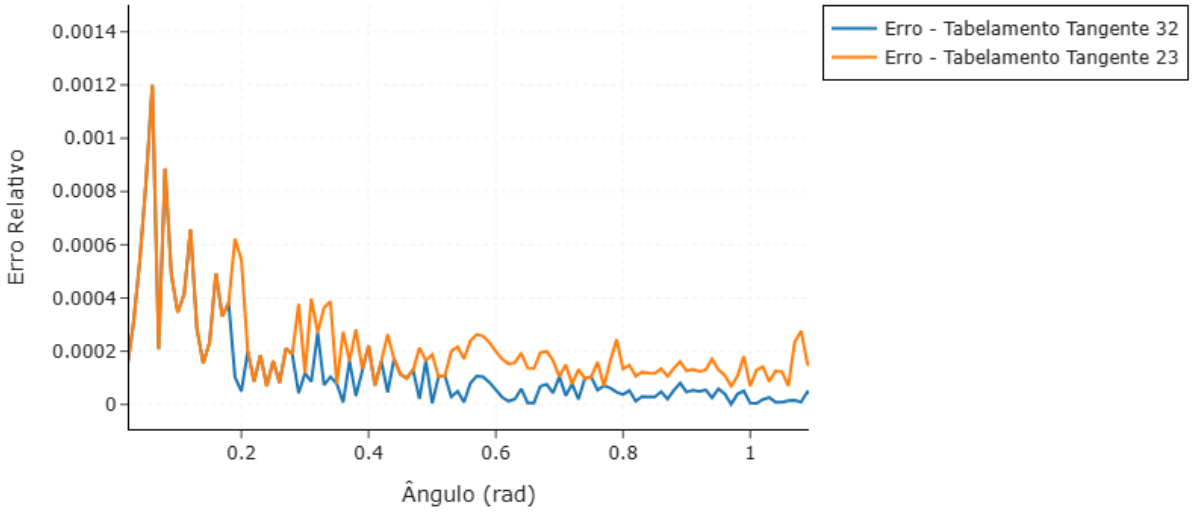

Figura 6.12 – Erro da função tangente através de tabelamento ampliado.

6.2.2 Arco Tangente, Tangente Hiperbólica e Sigmoide

Assim como no caso das aproximações por séries de Maclaurin, as funções arco tangente, tangente hiperbólica e sigmoide, que são funções que não apresentam periodicidade, foram testadas no intervalo $[-2, 2]$, com um passo de 0.01. É importante mencionar que, diferentemente das demais funções, a exponencial não é limitada. Ou seja, não é possível realizar uma aproximação de seu resultado a partir um valor finito, para valores maiores em módulo que um determinado valor de entrada. Sendo assim, não foi possível realizar a sua aproximação através do tabelamento.

Começando com a função arco tangente, é possível ver o seu resultado na Figura 6.13. Seu erro é relativamente uniforme, entre os intervalos $[-2, 0]$ e $[0, 2]$.

Já a função tangente hiperbólica pode ser vista na Figura 6.14. Ela tem um erro levemente maior no intervalo $[0, 2]$ em relação ao intervalo $[-2, 0]$, já que o seu valor absoluto é maior nesse intervalo. No entanto, o erro proporcional é bem similar entre os segmentos.

E por fim, para a função sigmoide, é possível ver seu erro absoluto na Figura 6.15. A mesma apresenta um erro bem similar à função tangente hiperbólica.

6.3 CORDIC

Para o processador com 32 bits, foi utilizado um número de iterações $N = 14$, já que, como explicado anteriormente, através de testes no processador foi determinado que a incerteza para esse método chega em um valor mínimo em 14 iterações, enquanto o valor

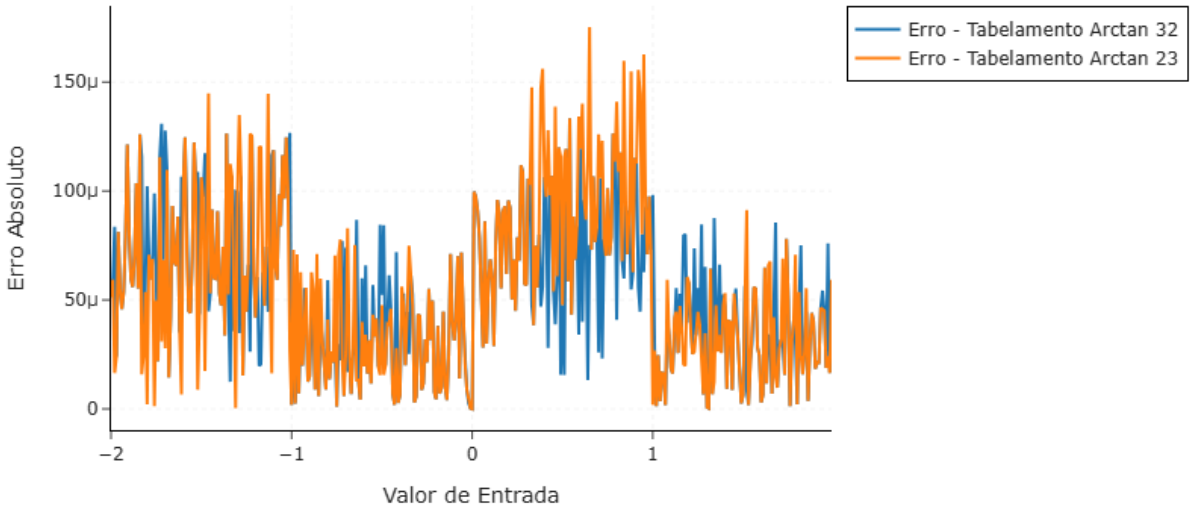

Figura 6.13 – Erro da função arco tangente através de tabelamento.

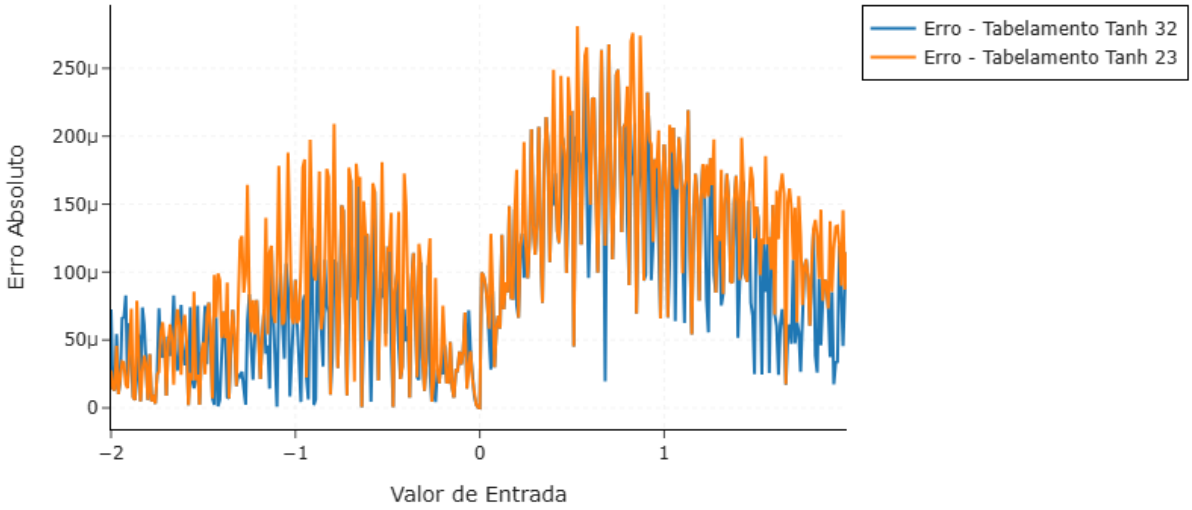

Figura 6.14 – Erro da função tangente hiperbólica através de tabelamento.

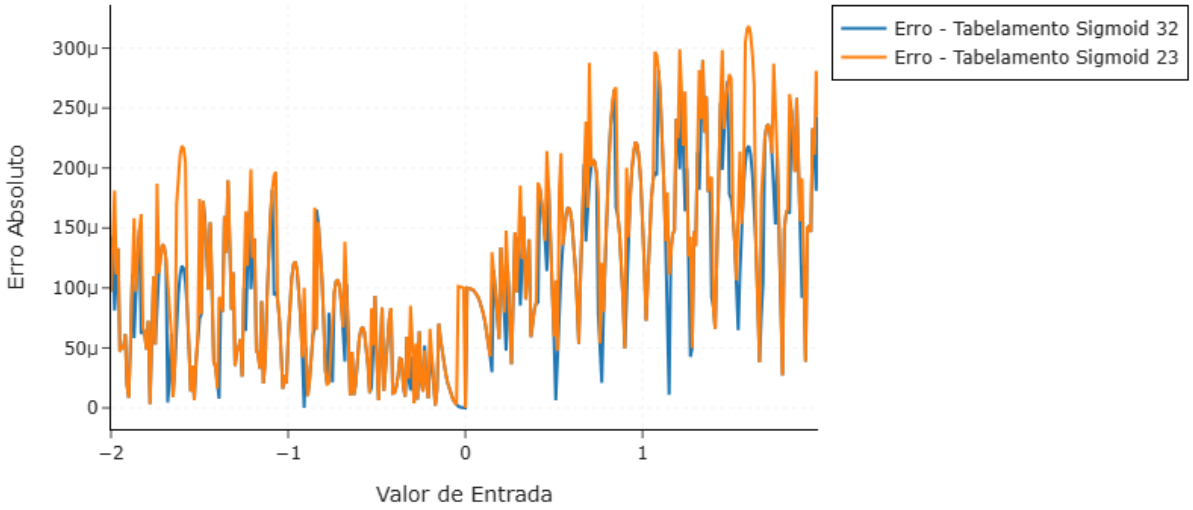

Figura 6.15 – Erro da função sigmoide através de tabelamento.

mínimo da incerteza é encontrado em $N = 11$ para o processador de 23 bits. Ademais, assim como nos casos anteriores, as funções periódicas foram avaliadas no intervalo $[0, 6.3]$, enquanto as funções não periódicas foram avaliadas no intervalo $[-2, 2]$.

6.3.1 Seno, Cosseno e Tangente

O Código 6.3 será utilizado como referência para a estrutura utilizada nas aplicações que utilizam o algoritmo CORDIC. Analisando o erro da função seno, vemos o comportamento mostrado na Figura 6.16. Na aproximação com o processador de 23 bits, nota-se picos em certos valores. Esses picos são explicados pela baixa precisão da representação. Próximo a certos ângulos, o acumulador, chamado de z_i no Código 6.3 fica no limiar entre ir para cima ou para baixo e, por causa de erros de arredondamento na representação em ponto flutuante de 23 bits, numa dada iteração, ele escolhe o sinal incorreto, fazendo uma rotação na direção contrária. O comportamento da aproximação em 32 bits pode ser vista na Figura 6.17.

Código 6.3 – Função Seno.

```
1 int N_iteracoes = 14;
2 float exp_2_neg[16] "2_-n.txt";
3 float arctan[16] "arctan_2_-n.txt";
4 float K = 0.607253;
5
6 void CORDIC_trig(float angulo){
7     angulo_normalizado = 0.0;
8     int quadrante = normalizador_de_angulo(angulo);
```

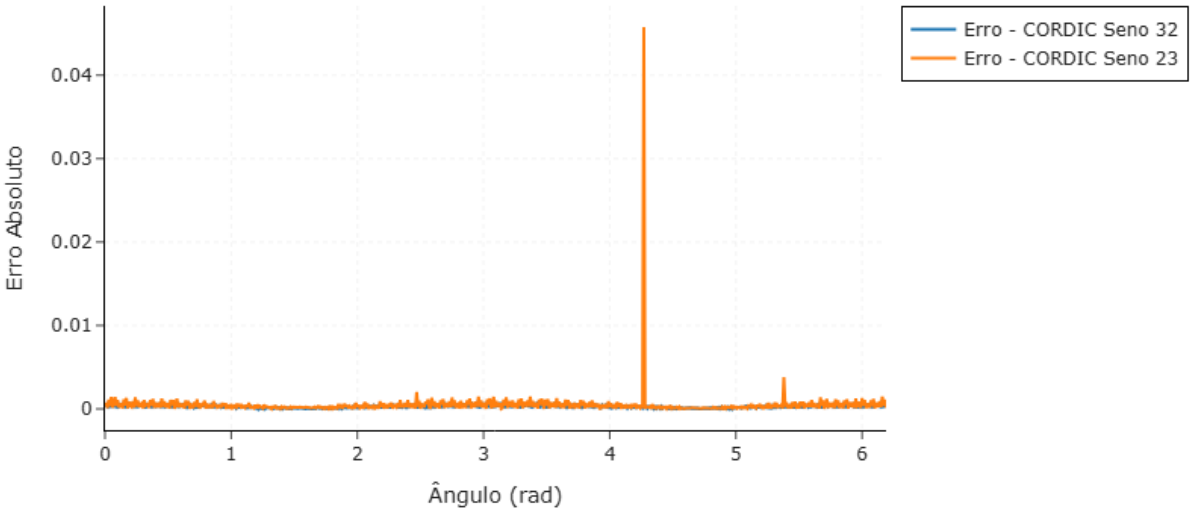

Figura 6.16 – Erro da função seno através do algoritmo CORDIC.

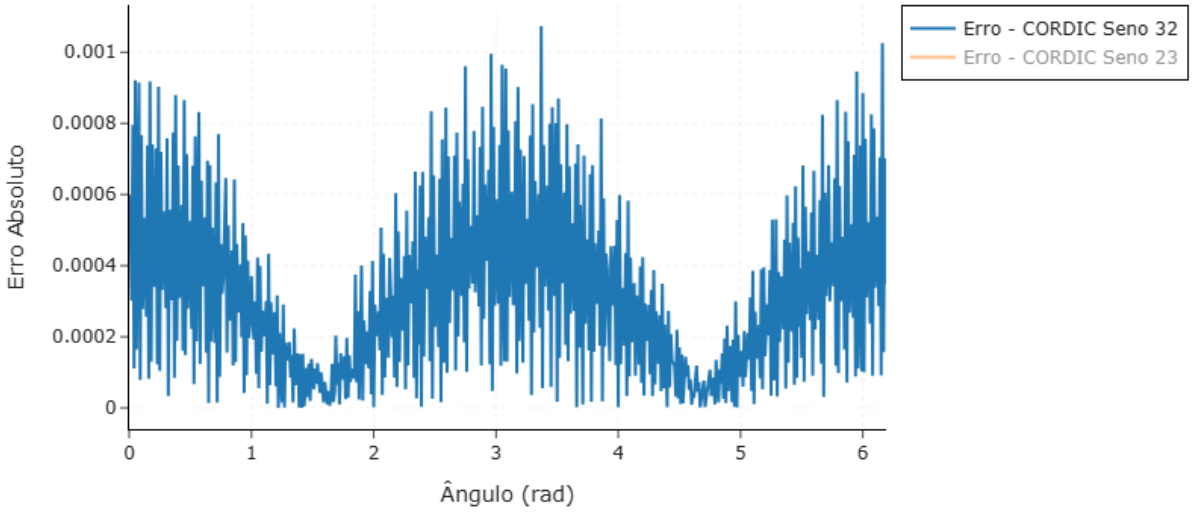

Figura 6.17 – Erro da função seno através do algoritmo CORDIC com 32 bits.

```
9
10      float x_i = K;
11      float x_i_anterior = 0.0;
12      float y_i = 0.0;
13      float z_i = angulo_normalizado;
14      int d_i = 0;
15
16      int k = 0;
17
18      while(k<N_iteracoes){
19          if(z_i >= 0) d_i = 1;
20          if(z_i < 0) d_i = -1;
21
22          x_i_anterior = x_i;
23
24          x_i = x_i - d_i * y_i * exp_2_neg[k];
25          y_i = y_i + d_i * x_i_anterior * exp_2_neg[k];
26          z_i = z_i - d_i * arctan[k];
27
28          k++;
29      }
30
31      if (quadrante == 3 || quadrante == 4) {
32          out_sen = -y_i;
33      }
34      else out_sen = y_i;
35  }
```

Enquanto o cosseno, com seu erro demonstrado na Figura 6.18, encontra-se o mesmo comportamento descrito na função seno. O erro de sua representação em 32 bits pode ser vista na Figura 6.19. Seu comportamento é extremamente similar ao visto na Figura 6.17, porém, defasado.

Já a função tangente tem seu erro visto na Figura 6.20, que foi representada da mesma forma que as aproximações por séries de Maclaurin e Tabelamento. A mesma apresenta picos de erro em valores em que a função diverge. É possível ver uma versão ampliada de seu erro em 6.21.

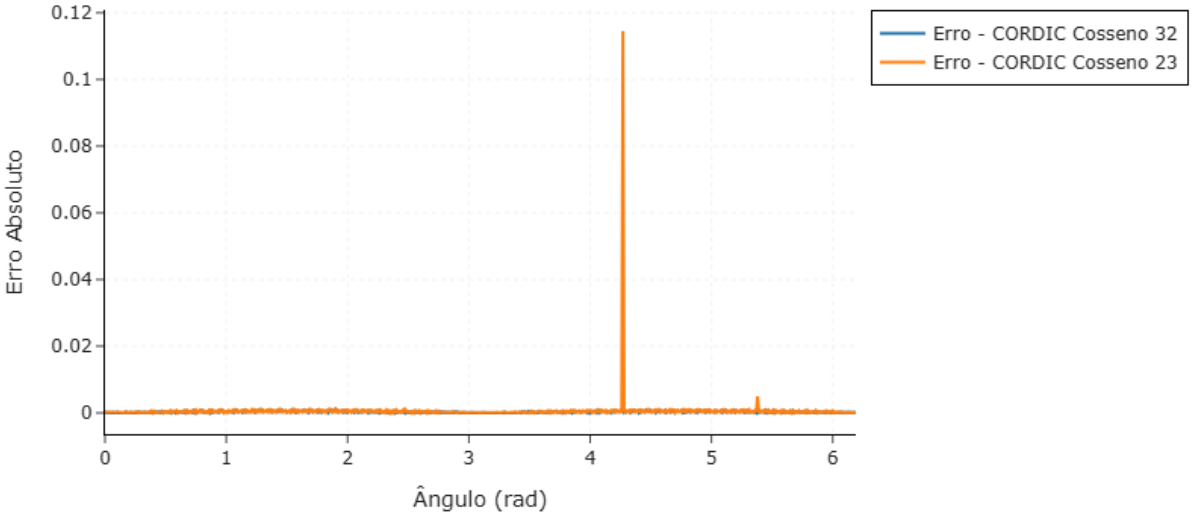

Figura 6.18 – Erro da função cosseno através do algoritmo CORDIC.

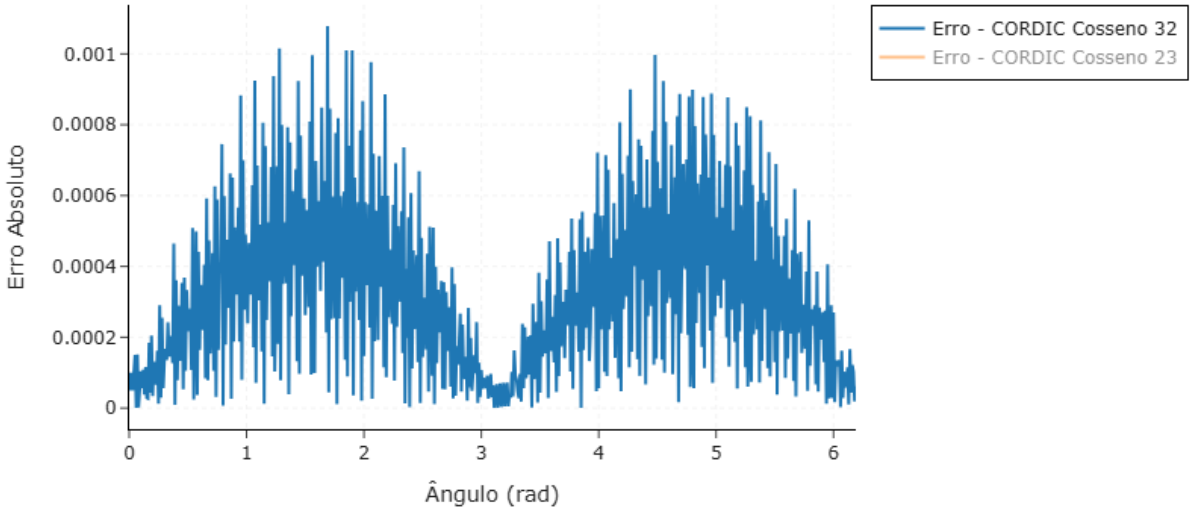

Figura 6.19 – Erro da função Cosseno através do algoritmo CORDIC com 32 bits.

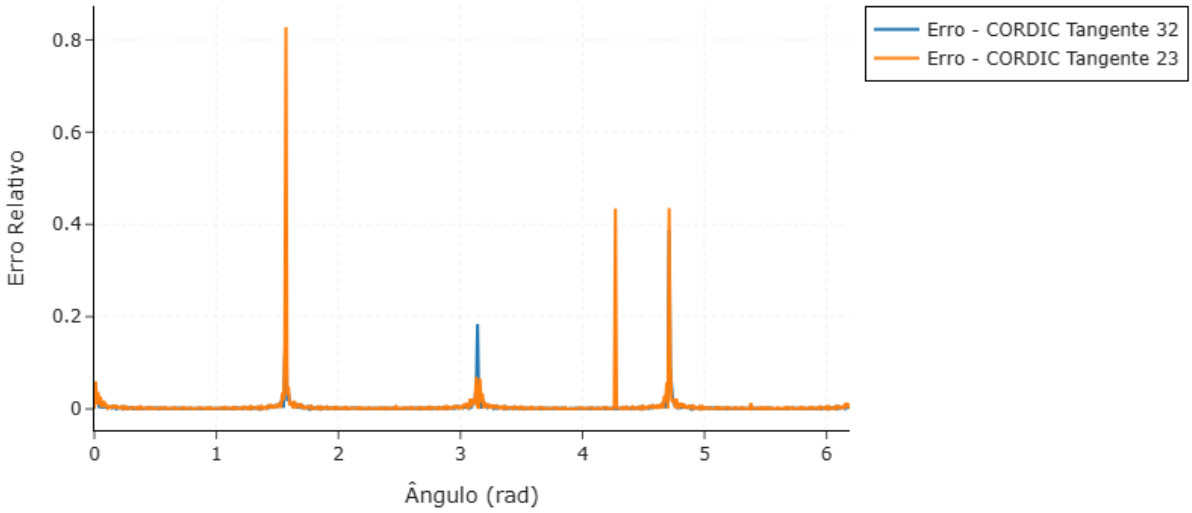

Figura 6.20 – Erro da função tangente através do algoritmo CORDIC.

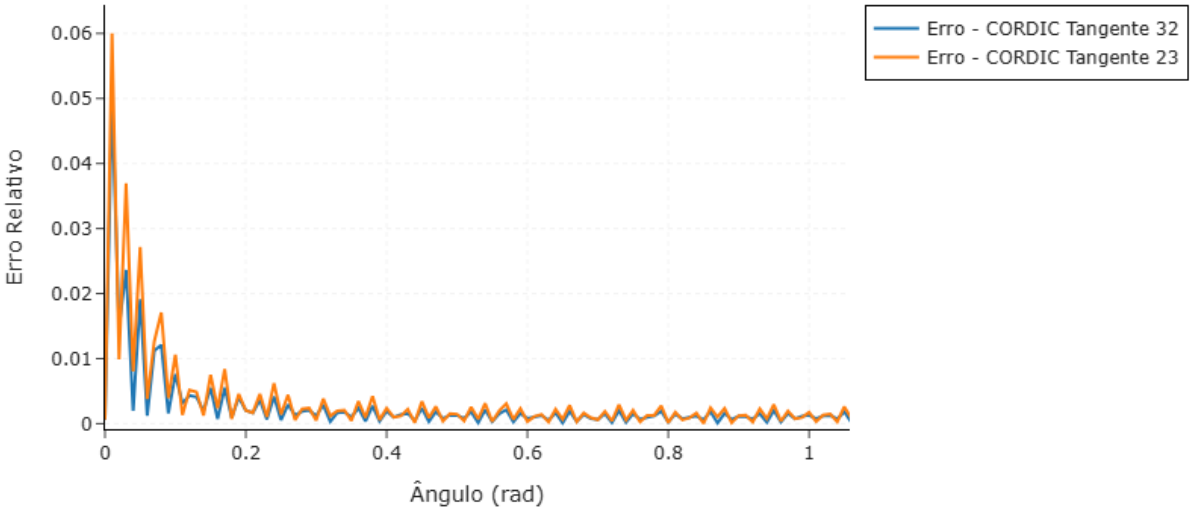

Figura 6.21 – Erro da função tangente através do algoritmo CORDIC ampliada.

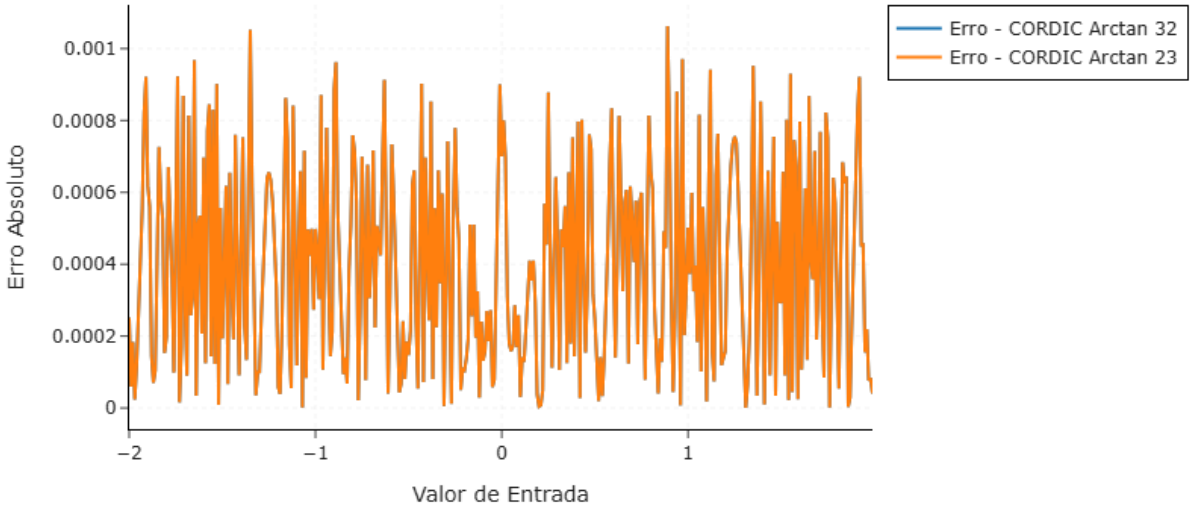

Figura 6.22 – Erro da função arco tangente através do algoritmo CORDIC.

6.3.2 Arco Tangente, Exponencial, Tangente Hiperbólica e Sigmoide

Cada uma das funções acima foi obtida de uma maneira levemente diferente, explicadas no Capítulo 5, sendo a função sigmoid obtida a partir da aproximação da função exponencial. A resposta da função arco tangente pode ser vista na Figura 6.22. Seu erro é uniforme no intervalo testado.

A função exponencial, de erro mostrado na figura 6.23, tem um comportamento único entre as demais. Seu erro muda de maneira considerável conforme o valor de entrada cresce. Por mais que o comportamento, de maneira geral, seja de aumento do erro, é possível notar que ainda há oscilações consideráveis no erro, explicadas pela proximidade de alguns valores do limiar de mudança de sinal para uma determinada iteração, como explicado no comportamento da função seno.

Já a função tangente hiperbólica é mostrada na Figura 6.24. Seu erro apresenta picos em torno dos valores ± 0.5 e ± 1, novamente, por causa da proximidade de certos valores nas iterações com o limiar de troca do sinal.

Por fim, tem-se a função sigmoide, na Figura 6.25. A mesma apresenta o menor erro de todas as funções aproximadas com o algoritmo CORDIC, por causa do uso de uma identidade que permite que a função sigmoide utilize apenas valores de entrada menores do que $\approx 0,3466$ na função exponencial, que apresenta um erro relativamente baixo para valores de entrada pequenos.

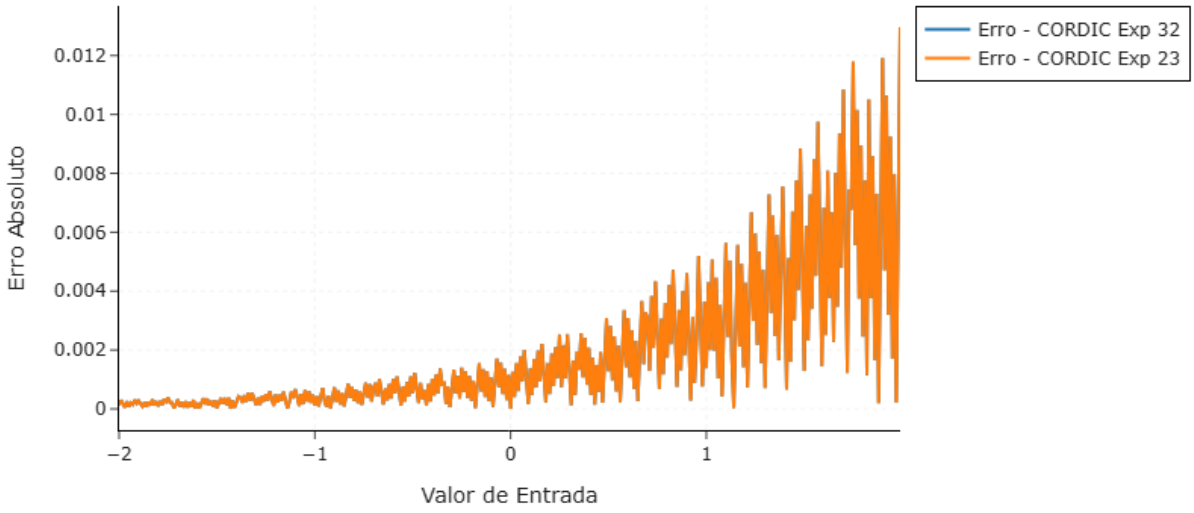

Figura 6.23 – Erro da função exponencial através do algoritmo CORDIC.

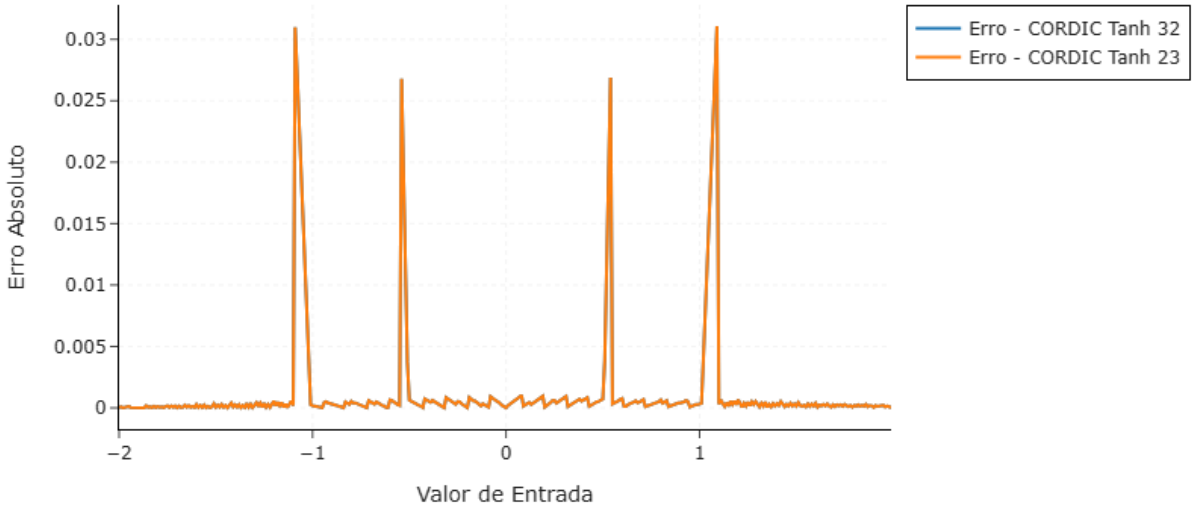

Figura 6.24 – Erro da função tangente hiperbólica através do algoritmo CORDIC.

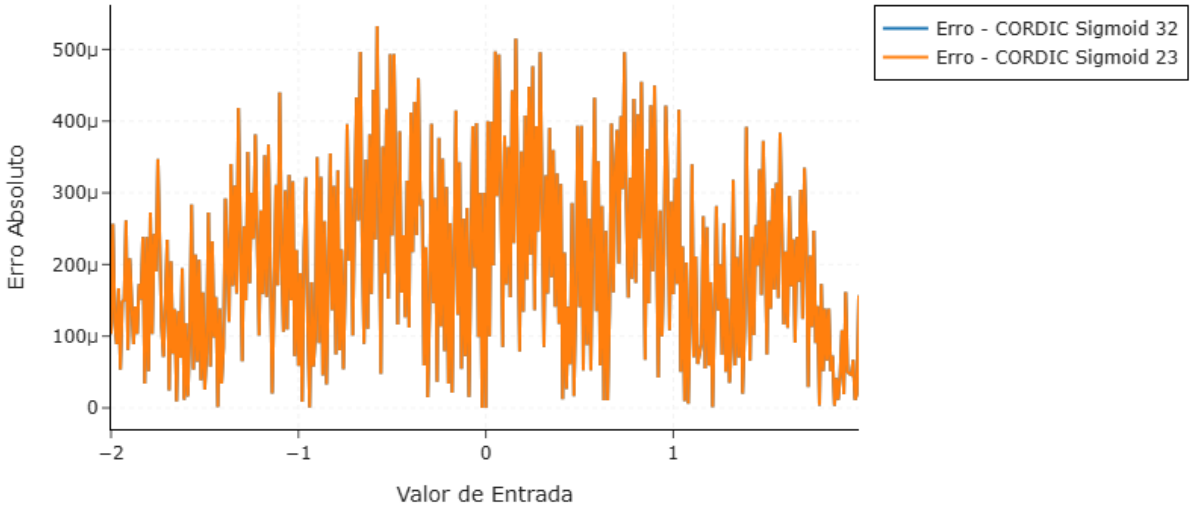

Figura 6.25 – Erro da função sigmoide através do algoritmo CORDIC.

6.4 COMPARAÇÃO DOS RESULTADOS

Os valores do erro médio de cada função utilizando séries de Maclaurin pode ser visto na Tabela 6.2, enquanto o número médio de ciclos de *clock* utilizados é mostrado na Tabela 6.3.

Tabela 6.2 – Erro médio com séries de Maclaurin.

Método	32	23
Seno	0.000050	0.000130
Cosseno	0.000050	0.001685
Tangente	0.000134	0.004729
Arco Tangente	0.000050	0.000390
Exponencial	0.000052	0.000785
Tangente Hiperbólica	0.000049	0.000560
Sigmoide	0.000049	0.000159

Tabela 6.3 – Número de ciclos de clock médio com séries de Maclaurin.

Método	32	23
Seno	232.744	164.357
Cosseno	248.956	175.853
Tangente	341.312	487.666
Arco Tangente	221.382	137.143
Exponencial	279.133	173.397
Tangente Hiperbólica	359.277	242.107
Sigmoide	265.286	166.910

Já os valores do erro médio de cada função utilizando o tabelamento pode ser visto na Tabela 6.4 e o número médio de ciclos de *clocks* são mostrados na Tabela 6.5.

Tabela 6.4 – Erro médio com tabelamento.

Método	32	23
Seno	0.000075	0.000113
Cosseno	0.000075	0.000114
Tangente	0.000152	0.001632
Arco Tangente	0.000053	0.000056
Exponencial	-	-
Tangente Hiperbólica	0.000085	0.000107
Sigmoide	0.000105	0.000122

Tabela 6.5 – Número de ciclos de clock médio com tabelamento.

Método	32	23
Seno	119.203	119.203
Cosseno	122.687	122.687
Tangente	238.377	238.377
Arco Tangente	51.824	51.824
Exponencial	-	-
Tangente Hiperbólica	49.289	49.289
Sigmoide	76.547	76.547

E por fim, é possível ver o erro médio de cada função sendo aproximada através do algoritmo *CORDIC* na Tabela 6.6 enquanto o número médio de ciclos de *clocks* é mostrado na Tabela 6.5.

Tabela 6.6 – Erro médio com *CORDIC*.

Método	32	23
Seno	0.000317	0.000486
Cosseno	0.000315	0.000587
Tangente	0.005031	0.007605
Arco Tangente	0.000401	0.000401
Exponencial	0.001890	0.001890
Tangente Hiperbólica	0.001295	0.001295
Sigmoide	0.000200	0.000200

Tabela 6.7 – Número de ciclos de clock médio com *CORDIC*.

Método	32	23
Seno	682.241	556.268
Cosseno	679.744	553.767
Tangente	685.007	560.026
Arco Tangente	1077.903	485.951
Exponencial	551.991	440.634
Tangente Hiperbólica	553.574	438.400
Sigmoide	567.991	456.634

7 CONCLUSÃO

Para facilitar a análise dos resultados apresentados no Capítulo 6, os resultados obtidos foram dispostos em dois gráficos, com o Gráfico 7.1 comparando os valores de erro médio para cada método e o Gráfico 7.2 comparando o número de ciclos de *clock* utilizados em cada método.

Ao analisar os resultados apresentados na Seção 6.4, é possível perceber que o algoritmo *CORDIC* não apresentou resultados satisfatórios, tanto em termos de precisão, quanto de consumo de recursos computacionais. Isso se dá por duas particularidades do processador utilizado e dos parâmetros de teste escolhidos. O método CORDIC, para o cálculo do seno como exemplo, precisa de aproximadamente 16 iterações para chegar em uma incerteza de ± 0.00005, já que seu cálculo é acrescido de $\pm 2^{-n}$ a cada iteração, como explicitado na Seção 5.1 e explicado por (ZHOU Z.L, 2023), o que o deixaria com uma precisão similar às aproximações através de séries de Maclaurin. No entanto, o erro introduzido com a soma e subtração entre valores extremamente pequenos representados em ponto flutuante com apenas 23 e 32 bits na *ULA* é ordens de magnitude maior do que o erro inerente ao algoritmo CORDIC com 16 iterações.

De fato, ao utilizar números menores de iterações, nota-se um aumento na precisão do método, com a representação de 32 bits tendo o menor erro médio para 14 iterações e a representação de 23 bits tendo menor erro médio em 11 iterações para as funções testadas. Por mais que o aumento arbitrário do número de bits da *ULA* na representação utilizada

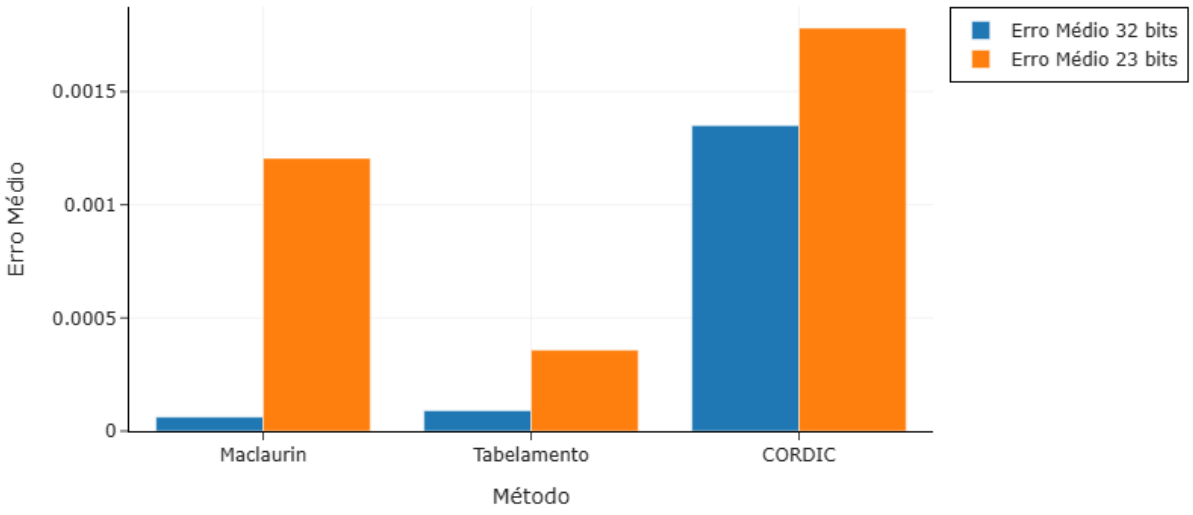

Figura 7.1 – Gráfico de barras do erro médio de cada método apresentado.

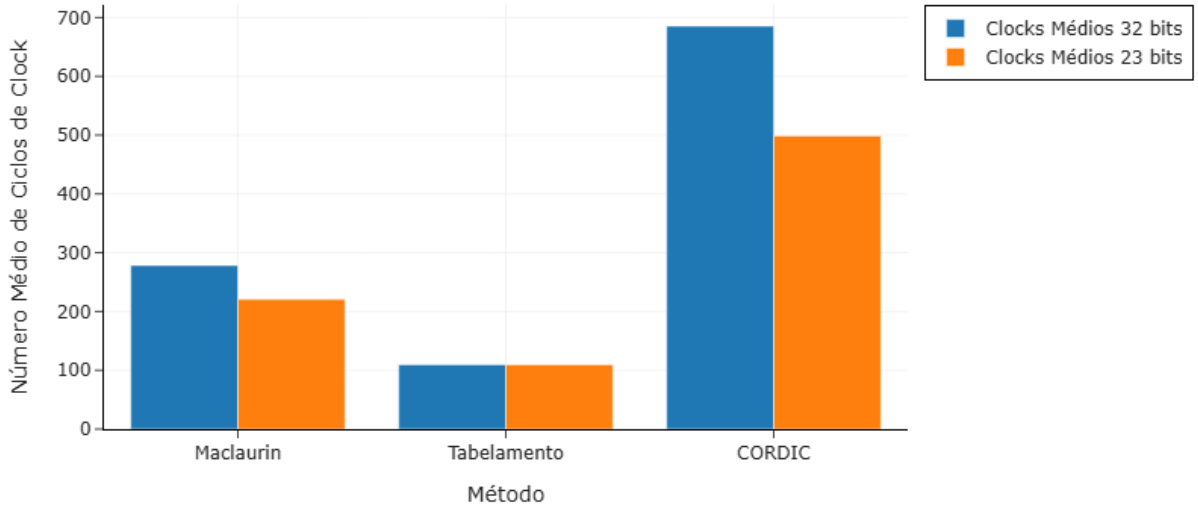

Figura 7.2 – Gráfico de barras do número de *clocks* médio de cada método apresentado.

seja uma possibilidade para aumentar a precisão, é importante enfatizar que todas as implementações aqui propostas têm como objetivo a busca de uma relação suficientemente proveitosa entre desempenho e precisão, fazendo com que o aumento do número de bits da representação e do número de iterações seja possível, mas na maioria dos casos imprático. Um método que pode aumentar a precisão, sem grandes impactos no número de ciclos de *clock* utilizados é a implementação do algoritmo CORDIC com inteiros quantizados, que pode ser explorado em trabalhos futuros.

Agora, partindo para a comparação entre o tabelamento e as séries de Maclaurin. Nota-se que cada método tem um contexto que torna-o mais adequado, com o tabelamento utilizando menos ciclos de clock em média, especialmente para as funções não periódicas, tendo em vista que o passo de normalização não é necessário nos mesmos, diferentemente das funções periódicas. Enquanto a implementação com séries de Maclaurin com 32 bits tem, na média, a maior precisão. É importante enfatizar que é possível alterar os resultados aqui vistos utilizando uma tolerância diferente ou um número diferente de 48 valores utilizados nas tabelas.

Não obstante, é possível dizer que, utilizando parâmetros idênticos aos utilizados no teste em questão, todas as funções cujo tabelamento foi realizado utilizando 23 bits tem melhor precisão e requerem menos ciclos de *clock* do que a implementação de 23 bits de séries de Maclaurin. É importante pontuar que o número médio de ciclos de clock utilizado no tabelamento é idêntico para 32 e 23 bits. Já as funções testadas com 32 bits utilizando séries de Maclaurin tiveram uma precisão maior do que as funções utilizando tabelamento com 32 bits, a troco de um número médio maior de ciclos de clock.

REFERÊNCIAS

AGARWAL R.P, P. K. P. S. **An Introduction to Complex Analysis**. [S.l.]: Springer, 2011.

AGUIAR R.G., N. V. M. F. H. H. Implmentação do algoritmo cordic para cálculo de seno e cosseno em fpga. **For Science**, 2020.

BOYCE W.E., D. R. M. D. **Elementary differential equations and boundary value problems.** [S.l.]: John Wiley & Sons, 2017., 2017.

BRIGGS I. LAD, Y. P. P. Implementation and synthesis of math library functions. **Cornell University**, 2023.

BURKOV, A. **The hundred-page machine learning book**. [S.l.]: Andriy Burkov, 2019.

CANNON J.W., F. W. K. R. . P. W. **Hyperbolic Geometry**. [S.l.]: MSRI Publications, 1997.

DRISCOLL T.A., B. R. **Fundamentals of numerical computation**. [S.l.]: Society for Industrial and Applied Mathematics, 2018.

ERCEGOVAC M.D., L. T. **Digital Arithmetic**. [S.l.]: Morgan Kaufmann Publishers, 2003.

GELFAND I.M., S. M. **Trigonometry**. [S.l.]: Birkhäuser Verlag, 2001.

KANTABUTRA, V. On hardware for computing exponential and trigonometric functions. **IEEE Transactions on Computers**, 1996.

KAPISCH E.B., A. M. F. L. S. L. Sapho - scalable-architecture processor for hardware optimization: an fpga customizable implementation approach.

KYURKCHIEV N., M. S. Sigmoid functions some approximation and modelling aspects. **ResearchGate**, 2015.

MOROZ L., S. V. The cordic method of calculating the exponential function. **Technical Transactions**, 2018.

PARRIS, R. Elementary functions and calculators. **Cacocntery Iundtbhns enf Deaduaethrs**, 2010.

PELLEGRINI, J. **Álgebra Linear**. [S.l.]: Aleph0, 2015.

PRESS W.H., T. S. V. W. F. B. **Numerical Recipes in C**. [S.l.]: Press Syndicate of the University of Cambridge, 2002.

SANTOS V.A.M., K. E. S. L. F. L. Implementação de circuitos aritméticos em ponto flutuante, utilizando formato com número de bits configurável. **Sociedade Brasileira de Automática**, 2019.

SEBAH P., G. X. Introduction on bernoulli's numbers. **http://numbers.computation.free.fr/**, 2002.

STEIN E.M., S. R. **Signals and systems**. [S.l.]: Princeton University Press, 2003.

STEWART J.D., C. D. W. S. **Cálculo: Volume 1**. [S.l.]: Cengage Learning, 2021.

VICCINI, L. **Otimizações para processador soft-core embarcado em FPGA visando implementação de métodos de reconstrução online de energia em aceleradores de partículas**. Dissertação (Mestrado) — UFJF, 2023.

ZHOU Z.L, V. D. Cordic algorithm and its applications. **Università degli Studi di Padova**, 2023.