

UNIVERSIDADE FEDERAL DO PARANÁ

MATEUS REBELLATO USSUI

SISTEMA DE PONTO FLUTUANTE COM PRECISÃO VARIÁVEL POSIT EM REDES  
NEURAIS INFORMADAS PELA FÍSICA (PINNS)

CURITIBA

2025

MATEUS REBELLATO USSUI

SISTEMA DE PONTO FLUTUANTE COM PRECISÃO VARIÁVEL POSIT EM REDES  
NEURAIS INFORMADAS PELA FÍSICA (PINNS)

Trabalho apresentado como requisito parcial  
para a obtenção do título de Bacharel em Mate-  
mática Industrial pela Universidade Federal do  
Paraná.

Orientador: Prof. Thiago de Oliveira Quinelato

CURITIBA

2025

*Dedico este trabalho*

*À minha mãe, Maristela, e ao meu irmão, Vitor, por me mostrarem, com exemplo e carinho, que o caminho do estudo é valioso, e por nunca deixarem de me apoiar, mesmo diante das dificuldades.*

*Aos meus amigos – Rhamon, Gabriella, Izabella, Hiago e tantos outros – por me escutarem, me aguentarem e estarem sempre por perto quando mais precisei.*

## **AGRADECIMENTOS**

A realização deste trabalho só foi possível graças ao apoio generoso de muitas pessoas, às quais expresso minha mais sincera gratidão.

Ao meu orientador, Prof. Dr. Thiago de Oliveira Quinelato, agradeço profundamente pela confiança depositada, pelas discussões técnicas sempre lúcidas e pela liberdade acadêmica que me permitiu explorar caminhos com autonomia e profundidade. Sua paciência – sempre invejável – e sua postura serena foram fundamentais ao longo de todo o processo.

Aos colegas do LabFluid e do curso de Matemática Industrial, em especial ao Larry e à Chloe, meu muito obrigado pela parceria e pelos diálogos que enriqueceram a vivência universitária como um todo. Estendo meus agradecimentos a todos os professores do curso, pela formação sólida, pela convivência generosa e pelos ensinamentos compartilhados ao longo dessa trajetória.

## RESUMO

Este trabalho tem como objetivo investigar a utilização do sistema numérico Posit em Redes Neurais Informadas pela Física (PINNs, do inglês *Physics-Informed Neural Networks*), com foco na substituição do tradicional sistema de representação em ponto flutuante IEEE 754. O sistema Posit, por sua natureza de precisão variável e distribuição não-uniforme, é particularmente promissor em cenários de baixa precisão, nos quais a eficiência computacional e a robustez numérica são críticas.

Inicialmente, viu-se necessária a implementação completa da estrutura Posit em Julia, incluindo codificação, decodificação, operações aritméticas fundamentais e operadores fundamentais, intitulada de PositPR. A biblioteca PositPR.jl foi construída com foco em extensibilidade e integração com fluxos de trabalho numéricos modernos, incluindo suporte a sobrecargas de operadores e funções matemáticas comuns.

Para viabilizar a aplicação prática dos Posits em redes neurais e métodos de aprendizado de máquina, foi necessária a adaptação de funções internas do ecossistema SciML, permitindo que o tipo Posit fosse reconhecido como um tipo numérico válido por bibliotecas como Lux.jl, NeuralPDE.jl e Optimization.jl. Essa integração incluiu o suporte à inicialização de pesos, operações com tensores e propagação direta e reversa com precisão arbitrária.

Os testes foram conduzidos em duas frentes: (i) análises da representação e da estabilidade numérica de funções e operações matemáticas sob diferentes configurações Posit; e (ii) aplicação direta em PINNs, com redes treinadas para resolver Equações Diferenciais Ordinárias (EDOs) e Equações Diferenciais Parciais (EDPs) clássicas.

**Palavras-chaves:** Posit; ponto flutuante; PINNs; aritmética de precisão variável; Julia; aprendizado de máquina; NeuralPDE.

## ABSTRACT

This work investigates the use of the Posit numerical system in Physics-Informed Neural Networks (PINNs), aiming to replace the traditional IEEE 754 floating-point format. Due to its variable precision and non-uniform distribution of representable numbers, the Posit system is particularly promising in low-precision scenarios in which computational efficiency and numerical robustness are critical.

Initially, it was deemed necessary to fully implement the Posit structure in Julia, including encoding, decoding, fundamental arithmetic operations, and basic operators, under the name PositPR. The PositPR.jl library was built with a focus on extensibility and integration with modern numerical workflows, including support for operator overloading and common mathematical functions.

To enable practical application of Posits in neural networks and machine learning methods, we adapted internal mechanisms within the SciML ecosystem, allowing the Posit type to be recognized as a valid numeric type in libraries such as Lux.jl, NeuralPDE.jl, and Optimization.jl. This integration includes support for weight initialization, tensor operations, and both forward and backward propagation in arbitrary precision.

Experiments were carried out in two main directions: (i) analysis of numerical representation and stability of mathematical operations under different Posit configurations; and (ii) direct application to PINNs, training networks to solve classical ODEs and PDEs.

**Key-words:** Posit; floating-point; PINNs; variable-precision arithmetic; Julia; machine learning; NeuralPDE.

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	7
<b>2</b>	<b>SISTEMA NUMÉRICO POSIT</b>	10
2.1	ESTRUTURA	10
2.2	VANTAGENS	11
2.3	BIBLIOTECA POSITPR.JL	12
<b>3</b>	<b>SISTEMAS NUMÉRICOS DE BAIXA PRECISÃO EM REDES NEURAIS</b>	15
3.1	PRECISÃO ADAPTATIVA NO TREINAMENTO	16
3.2	POSIT COMO ALTERNATIVA	16
<b>4</b>	<b>APLICAÇÃO DE POSIT EM PINNS</b>	18
4.1	PINNS: REDES NEURAIS INFORMADAS PELA FÍSICA	18
4.2	ADAPTAÇÃO DE BIBLIOTECAS	18
4.3	GERADOR DE NÚMEROS ALEATÓRIOS EM POSIT	20
4.3.1	Distribuição Uniforme	20
4.3.2	Distribuição Normal	21
<b>5</b>	<b>ANÁLISE EXPERIMENTAL E RESULTADOS</b>	22
5.1	AVALIAÇÃO DE PRECISÃO NUMÉRICA DE POSITPR.JL	22
5.1.1	Precisão e Densidade de Representações Numéricas	22
5.1.2	Operações Aritméticas Elementares	24
5.1.3	Avaliação de Funções	26
5.2	APLICAÇÕES EM REDES NEURAIS	34
5.2.1	Inicialização de Redes (Lux.jl)	34
5.2.2	Treinamento da rede	35
5.2.2.1	Sem NeuralPDE.jl	36
5.2.2.2	Com NeuralPDE.jl	38
5.2.3	Performance da rede	39
<b>6</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS</b>	52
	<b>REFERÊNCIAS</b>	54

## 1 INTRODUÇÃO

A proposta deste projeto consiste em explorar como o uso do sistema numérico Posit, como alternativa ao padrão tradicional IEEE-754 de ponto flutuante (Gustafson; Yonemoto, 2017), na definição da arquitetura e na etapa de treinamento de modelos de redes neurais, pode contribuir para melhorar aspectos computacionais importantes – como uso de memória e eficiência energética (Lu et al., 2019). O foco estará na aplicação do Posit na inicialização de pesos e vieses da rede no contexto das Redes Neurais Informadas pela Física (PINNs, do inglês *Physics-Informed Neural Networks*), que incorporam restrições físicas na construção da função de perda.

O crescimento do interesse por sistemas numéricos alternativos tem sido acompanhado por um número crescente de publicações e eventos dedicados ao tema. Dentre eles, destaca-se o *Conference on Next Generation Arithmetic* (CoNGA)<sup>1</sup>. Voltado ao desenvolvimento de novas abordagens para aritmética computacional, o evento destaca-se por reunir pesquisas que propõem soluções mais rápidas, precisas e energeticamente eficientes do que o padrão tradicional IEEE-754. Com foco em demandas computacionais modernas, como inteligência artificial (IA) e computação de alto desempenho (HPC), o CoNGA tem consolidado o Posit como um formato emergente não proprietário.

Contudo, apesar dos avanços no estudo do sistema Posit e de sua promessa como substituto eficiente para os formatos tradicionais de ponto flutuante, até o momento não existia uma biblioteca de código aberto voltada à experimentação prática com Posits em testes numéricos. Diante dessa lacuna, este trabalho propõe e apresenta a implementação da biblioteca PositPR.jl, uma infraestrutura desenvolvida em Julia, que oferece suporte completo ao sistema Posit com operações aritméticas fundamentais, funções e uma integração direta com frameworks modernos de aprendizado de máquina, como Lux.jl e NeuralPDE.jl. Essa contribuição viabiliza, pela primeira vez de forma aberta e reproduzível, a realização de experimentos numéricos usando Posit.

O sistema numérico Posit oferece uma representação de ponto flutuante com precisão variável, maior densidade de representação para números próximos de 1 e melhor aproveitamento de bits do que os formatos tradicionais (Gustafson; Yonemoto, 2017). Estudos recentes demonstram que redes neurais podem ser treinadas com precisão reduzida em Posit (8 ou 16 bits) com resultados comparáveis – e, por vezes, superiores – à precisão tradicional de 32 bits em números de ponto flutuante de precisão fixa, sem perda significativa de acurácia (Lu et al., 2019).

---

<sup>1</sup> <https://posithub.org/events>

O treinamento de Redes Neurais Profundas (DNNs, do inglês *Deep Neural Networks*) apresenta altos custos computacionais, exigindo grande capacidade de memória e um número massivo de operações aritméticas. Essa complexidade torna inviável, ou ao menos muito custoso, o uso de DNNs em dispositivos com recursos limitados ou em contextos que exigem múltiplas simulações, como problemas físicos complexos (Lu et al., 2021). Assim, ao utilizar uma representação compacta como o Posit de 16 bits para representar as variáveis internas do modelo (pesos e vieses), espera-se uma redução significativa na complexidade aritmética, no uso de memória e na propagação de erros de arredondamento ao longo da rede.

A biblioteca PositNN (Raposo et al., 2021), construída em C++ com suporte a diferentes precisões e acúmulos exatos via *quires*, demonstrou a viabilidade de treinar Redes Neurais Convolucionais (CNNs, do inglês *Convolutional Neural Networks*) com números Posit entre 8 e 12 bits. Os resultados obtidos mostraram desempenho comparável ao uso de Floats de 32 bits, inclusive em tarefas desafiadoras como a classificação do conjunto Fashion MNIST.

Outros estudos, como o Deep Positron (Carmichael et al., 2019), reforçam o potencial do Posit para inferência com baixa precisão em FPGA (do inglês *Field-Programmable Gate Array*), evidenciando ganhos significativos em energia e desempenho quando comparado a formatos tradicionais como ponto flutuante e ponto fixo.

Apesar dessas contribuições, não há, até o presente momento, estudos que explorem o uso de representações numéricas alternativas – como o Posit – no contexto específico das PINNs, cuja estrutura exige alto rigor numérico para equilibrar a aprendizagem de dados com o cumprimento de equações diferenciais.

Neste projeto, propõe-se justamente investigar essa lacuna, integrar a biblioteca PositPR.jl ao ecossistema Julia e avaliar os impactos dessa substituição numérica no desempenho e eficiência das PINNs.

O primeiro passo é o estudo da integração da biblioteca PositPR.jl com componentes do ecossistema de aprendizado de máquina e resolução de Equações Diferenciais Parciais (EDPs) por PINNs, como Flux.jl (Innes et al., 2018; Innes, 2018), Lux.jl (Pal, 2023a,b), Optimization.jl (Dixit; Rackauckas, 2023) e NeuralPDE.jl (Zubov et al., 2021). É necessário fazer uma adaptação de PositPR.jl, que possui todas as operações aritméticas implementadas, com os componentes internos das bibliotecas-alvo; neste caso, a inicialização dos pesos e vieses. Assim, as operações feitas na rede serão todas em Posit para diferentes configurações de Posit, parametrizado por duas variáveis (*ps* e *es*, que definem, respectivamente, o número de bits usados para cada posit e, desses, até quantos podem ser usados para armazenar o expoente do número).

Após a adaptação, testes serão realizados com problemas padrão resolvidos

por PINNs (por exemplo, a equação do calor), utilizando diferentes formatos numéricos. Serão avaliadas diferentes métricas para os resultados obtidos, identificando os benefícios e limitações do uso de Posit. Por fim, o código será organizado em formato reutilizável e a documentação do projeto será preparada para possibilitar sua continuidade ([código disponível](#)).

Este trabalho assume familiaridade prévia com fundamentos de representação numérica, em particular o modelo de ponto flutuante IEEE-754, bem como noções elementares de aprendizado de máquina supervisionado. Ainda que não seja necessário um conhecimento profundo sobre as técnicas de PINNs, espera-se que o leitor esteja confortável com conceitos básicos de redes neurais, como arquitetura de camadas, propagação direta e retropropagação do erro.

## 2 SISTEMA NUMÉRICO POSIT

O sistema numérico Posit representa uma alternativa ao padrão IEEE 754 para aritmética de ponto flutuante (IEEE, 2019), buscando superar suas limitações quanto à precisão fixa, desperdício de representações com valores excepcionais ( $\text{NaN}$  e  $\pm\infty$ ) e comportamento inconsistente em arredondamentos e operações com zero. Ao contrário dos Floats, os Posits oferecem precisão variável e uma representação mais eficiente de números reais com poucos bits.

## 2.1 ESTRUTURA

O sistema de ponto flutuante tradicional, padronizado pela IEEE-754, representa números reais em uma estrutura fixa de bits, conforme ilustrado na Figura 1. No formato de 32 bits (Float32), temos: 1 bit de sinal, indicando o sinal do número; 8 bits de expoente, com viés (bias) aplicado; 23 bits de mantissa, representando a parte fracionária do número normalizado.

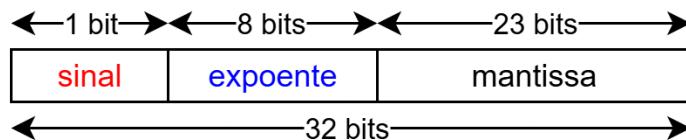


FIGURA 1 – Estrutura dos bits no padrão IEEE-754 (32 bits).

A fórmula geral de representação de um número em IEEE-754 é dada por:

$$x_{\text{Float}\{\text{fs}\}} = (-1)^s \cdot 2^e \cdot \left(1 + \sum_{k=1}^N a_k 2^{-k}\right),$$

onde  $e$  é o expoente ajustado pelo viés, e os  $a_k$  são os bits da mantissa.

Por outro lado, o sistema Posit propõe uma alternativa com estrutura variável e adaptativa. O número Posit é construído com base nos parâmetros  $ps$  (número total de bits) e  $es$  (quantidade máxima de bits do expoente). A fórmula de representação é:

$$x_{\text{Posit}\{\text{ps,es}\}} = (-1)^s \cdot \text{useed}^k \cdot 2^e \cdot \left(1 + \sum_{k=1}^N a_k 2^{-k}\right), \quad \text{onde } \text{useed} = 2^{2^{\text{es}}}.$$

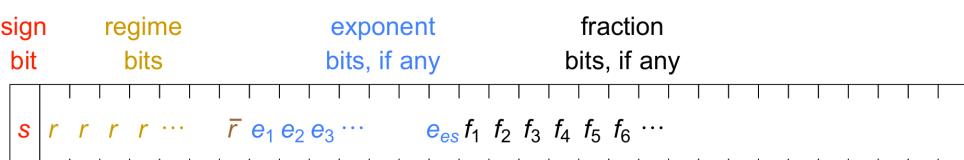


FIGURA 2 – Estrutura dos bits no Sistema Posit. Fonte: adaptado de Gustafson e Yonemoto (2017).

A codificação dos bits em um Posit é feita da esquerda para a direita, começando pelo bit de sinal  $s$ , seguido por uma sequência de bits iguais denominada *regime*, que termina ao encontrar um bit de valor contrário (essa sequência determina o valor de  $k$ , sendo positiva se inicia com bits 1 e negativa se inicia com bits 0). Em seguida vêm os bits do expoente, em número de até  $es$  bits, e por fim os bits restantes são destinados à fração, que representa os dígitos significativos do número.

Diferente do IEEE 754, os limites entre as partes não são fixos, o que permite ao Posit adaptar dinamicamente o uso dos bits disponíveis conforme o valor representado.

## 2.2 VANTAGENS

O sistema Posit apresenta diversas características que o diferenciam do padrão IEEE 754. Uma das mais notáveis é sua precisão variável: números próximos de zero tendem a alocar mais bits para a fração, o que proporciona maior fidelidade nessa região da reta real – justamente onde ocorre a maioria das operações numéricas em aprendizado de máquina e computação científica. Além disso, o formato Posit possui uma faixa dinâmica consideravelmente maior que a dos sistemas de ponto flutuante tradicionais com o mesmo número de bits. Isso é possível devido à estrutura do campo de regime, cujo crescimento é exponencial com base no parâmetro  $es$ , controlado por  $useed = 2^{2^{es}}$ . O campo de regime funciona como um “super-expoente”, permitindo representar números com ordens de magnitude muito altas ou muito baixas utilizando poucos bits.

Mais do que apenas ampliar o alcance dinâmico, o regime também colabora para o aumento de precisão local, pois, à medida que a magnitude do número representado se afasta da unidade, o regime consome mais bits e reduz o espaço para o expoente e para a fração. No entanto, ao redor da unidade – onde se concentram a maioria das ativações, pesos e gradientes em redes neurais – a quantidade de bits disponíveis para a fração é maximizada, o que implica maior resolução.

Como ilustrado por Gustafson (Gustafson; Yonemoto, 2017), existe um ponto ótimo de precisão quando se utilizam valores de  $es = 3$  ou  $es = 4$  para representar quantidades na ordem de milhares. Isso acontece porque o regime permite representar potências de dois de forma compacta até atingir um padrão de codificação mínimo (como o par de bits 10), momento a partir do qual o expoente passa a atuar com granularidade mais uniforme. Essa propriedade garante que valores como  $10^4$  possam ser representados exatamente, mesmo com apenas 16 bits disponíveis – algo que não é possível com o Float16, por exemplo.

Outro aspecto importante é a simplicidade no tratamento de arredondamento: há apenas uma regra adotada, o arredondamento para o mais próximo, com empate

indo para o bit par (*tie to even*). Além disso, o sistema Posit elimina a complexidade associada a valores excepcionais presentes no padrão IEEE 754. Enquanto o Float tradicional reserva inúmeras *bitstrings* para representar NaNs (valores indefinidos) e possui distinções artificiais como  $+0$  e  $-0$ , o Posit define apenas duas exceções: a *bitstring* composta inteiramente por zeros representa o número zero, e aquela com um 1 seguido apenas por zeros representa o infinito projetivo ( $\pm\infty$ ). Todas as demais combinações de bits codificam números reais válidos, o que não apenas aumenta a eficiência no uso de representações como também evita ambiguidade semântica durante os cálculos.

Em resumo, o formato Posit alia alta precisão relativa onde ela é mais necessária a uma cobertura dinâmica excepcional, eliminando redundâncias de codificação e garantindo que quase todas as *bitstrings* sejam mapeadas para números representáveis. Isso o torna uma alternativa promissora para computação em baixa precisão, especialmente em contextos sensíveis à consistência e à expressividade numérica, como o treinamento de redes neurais profundas.

### 2.3 BIBLIOTECA POSITPR.JL

Durante o desenvolvimento deste trabalho, constatou-se a ausência de uma biblioteca em Julia que permitisse realizar operações numéricas diretamente sobre o sistema Posit com precisão arbitrária. A única referência disponível foi a SoftPosit.jl, que oferecia apenas funções de conversão entre formatos Posit e ponto flutuante, sem suporte à aritmética básica.

Diante desse cenário, iniciou-se o desenvolvimento da biblioteca PositPR.jl, sendo o sufixo PR uma homenagem à Universidade Federal do Paraná (UFPR), onde o projeto foi idealizado. O trabalho teve início em um Programa de Voluntariado Acadêmico (PVA) e, posteriormente, contou com o apoio de uma bolsa de Iniciação Científica concedida pelo INCTMat. O foco do projeto foi o estudo da representação dos números reais em computadores e, mais especificamente, de formatos alternativos ao padrão IEEE-754, culminando na implementação prática do sistema Posit.

A biblioteca PositPR.jl foi construída para permitir a criação de instâncias Posit com parâmetros arbitrários de número total de bits (`ps`) e tamanho máximo do campo de expoente (`es`), por meio da estrutura paramétrica `Posit{ps,es}`. Essa abordagem não se limita a configurações fixas como  $(8, 1)$  ou  $(16, 1)$ , possibilitando simulações em diferentes níveis de granularidade e alcance dinâmico.

Cada número Posit gerado pela biblioteca é instanciado com seus campos internos explicitamente armazenados, permitindo acesso detalhado às etapas de codificação, decodificação e aritmética. A estrutura principal é definida da seguinte

forma:

```
struct Posit{ps, es}<:Number
    s::Int      #bit de sinal
    sn::Int     #sinal numerico (+-1)
    k::Int      #valor do regime
    rs::Int     #tamanho do campo de regime
    ers::Int    #tamanho do campo de expoente
    e::Int      #valor do expoente
    fi::String  #bits da fracao como string
    fs::Int     #tamanho da fracao
end
```

Além da representação estrutural, todas as comparações fundamentais entre números foram implementadas, incluindo igualdade, diferença, menor, maior e suas variações não estritas. Essas operações foram fundamentais para viabilizar a implementação completa da aritmética Posit e funções descritas na Tabela 1.

TABELA 1 – Principais funções implementadas na biblioteca PositPR.jl

Função	Descrição
==, !=, <, >, <=, >=	Comparações entre Posits
one, zero, isone, iszero	Constantes e verificações
+, -, *, /	Aritmética básica entre Posits
^, sqrt	Potenciação e radiciação
abs, round, floor, mod	Operações auxiliares
convert, Int	Conversão entre Posit e tipos nativos
x2Posit, Posit2x	Conversão entre Posit e Float64
Posit2Posit	Conversão entre dois tipos Posit com diferentes parâmetros
eps, prevPosit, nextPosit	Precisão e vizinhança numérica
exp, log, ln	Funções exponenciais e logarítmicas
sin, cos, tan	Funções trigonométricas
», <	Shift lógico para otimizações específicas
rand, randn	Geração de números aleatórios em distribuições uniforme e normal

Vale ressaltar que a implementação da PositPR.jl não se limita à definição de uma nova estrutura de dados, mas consiste na construção completa de um sistema numérico funcional partindo do zero, sem recorrer a operações nativas de ponto flutuante nem a bibliotecas externas especializadas em aritmética Posit. Trata-se de um trabalho de alta complexidade, que exigiu compreender profundamente a estrutura lógica dos números Posit – o comportamento do regime, expoente e fração – e traduzi-la em algoritmos robustos, consistentes e genéricos para quaisquer valores de ps e es. Isso incluiu o desenvolvimento das etapas de codificação e decodificação, comparações

lógicas e toda a aritmética de precisão variável. Cada operação principal exigiu o encadeamento de diversas funções auxiliares, projetadas para respeitar a semântica da representação Posit. A biblioteca também foi estruturada com modularidade, permitindo a extensão para testes, geração de números aleatórios e integração com outras ferramentas numéricas. O domínio sobre a lógica interna desse sistema e a capacidade de implementar suas operações fundamentais de forma autônoma representa uma contribuição significativa no estudo e experimentação de representações numéricas alternativas.

Com essas funcionalidades, é possível realizar testes numéricos utilizando diferentes configurações de Posit, investigando como a variação dos parâmetros  $ps$  e  $es$  impacta na precisão dos resultados.

### 3 SISTEMAS NUMÉRICOS DE BAIXA PRECISÃO EM REDES NEURAIS

O crescimento exponencial do uso de redes neurais profundas tem trazido consigo um aumento expressivo na demanda por recursos computacionais. Modelos modernos, como transformadores e redes convolucionais profundas, exigem bilhões de parâmetros e realizam trilhões de operações durante o treinamento. Esse cenário impulsiona a busca por técnicas que reduzam o custo computacional sem comprometer o desempenho dos modelos.

Uma das estratégias mais promissoras nesse contexto é a adoção de sistemas numéricos de baixa precisão. Ao representar números com menos bits, é possível diminuir o uso de memória, aumentar a velocidade de execução (em *hardware* especializado) e reduzir o consumo de energia – mas, principalmente, manter a precisão dos cálculos essenciais ao aprendizado, como as atualizações dos pesos e dos vieses.

No entanto, a simples redução do número de bits pode comprometer drasticamente a estabilidade do treinamento e a qualidade da inferência. Representações como `Float8` e `Int8`, embora eficientes em termos de espaço, apresentam limitações sérias quanto à faixa dinâmica, à resolução em torno de zero e à capacidade de representar gradientes sutis, elementos centrais no processo de aprendizado.

Para manter a integridade do fluxo de informação durante o treinamento, é necessário que a representação numérica preserve com qualidade os valores pequenos, frequentemente encontrados na inicialização de pesos e na magnitude das atualizações, o que exige uma alta densidade de representações próximas de zero.

Além disso, é desejável que o sistema numérico ofereça precisão variável, isto é, que a mantissa se ajuste dinamicamente ao valor representado. Com isso, é possível alocar mais bits para representar números que exigem maior precisão relativa (como aqueles próximos da unidade) e menos bits para valores extremos, otimizando o uso da capacidade representacional (Jost, 2021). Outro aspecto crítico é a robustez das operações aritméticas: a consistência e a reproduzibilidade dos cálculos, mesmo com baixa precisão, são essenciais para garantir a estabilidade do aprendizado.

Portanto, uma representação numérica de baixa precisão que reúna essas propriedades permite manter a integridade dos resultados computacionais em redes neurais profundas, ao mesmo tempo em que reduz significativamente o custo de armazenamento e o consumo energético, favorecendo sua aplicação em dispositivos com restrições de *hardware*.

### 3.1 PRECISÃO ADAPTATIVA NO TREINAMENTO

Redes neurais profundas vêm sendo amplamente exploradas sob diferentes regimes de precisão numérica, impulsionadas pelo desejo de reduzir custos computacionais e acelerar o treinamento e a inferência. No entanto, a escolha da precisão tem implicações diretas sobre a estabilidade, convergência e desempenho dos modelos, especialmente durante o treinamento.

Enquanto a inferência é geralmente robusta a representações de baixa precisão – como Float16, Int8 ou até mesmo formatos com menos de 8 bits – o treinamento requer maior cuidado. O processo de retropropagação depende da preservação precisa de gradientes, que frequentemente assumem valores muito pequenos nas primeiras épocas. Erros de arredondamento podem interromper o fluxo de informação entre as camadas, levando à estagnação do aprendizado ou divergência (Lu et al., 2019).

Para lidar com esse desafio, uma das estratégias mais utilizadas é o uso de precisão mista (*mixed precision*), com diferentes partes do treinamento utilizando formatos numéricos distintos. Como demonstrado por Raposo et al. (2021), os gradientes e os erros propagados requerem representações com ampla faixa dinâmica e boa resolução em torno de zero, enquanto as ativações intermediárias e os pesos, após estabilização, podem ser armazenados com menos bits. Essa abordagem permite acelerar o treinamento mantendo a precisão onde ela é mais sensível.

Outro conceito relevante é o do *warmup*, em que as primeiras iterações do treinamento utilizam maior precisão (como Float32) para permitir a estabilização das distribuições estatísticas dentro da rede. Após essa fase, é possível migrar para representações mais compactas sem comprometer a convergência (Lu et al., 2019). Essa mudança pode ser feita de forma gradual ou por blocos, respeitando a evolução das métricas de perda e a sensibilidade dos parâmetros.

Dessa forma, estratégias adaptativas de precisão têm se mostrado uma via promissora para combinar eficiência computacional com desempenho robusto, tanto em ambientes restritos (como sistemas embarcados, por exemplo) quanto em grandes treinamentos distribuídos. A próxima seção apresenta uma proposta baseada no sistema numérico Posit, que oferece flexibilidade para aplicar essas estratégias com granularidade variável e suporte completo à aritmética de precisão arbitrária.

### 3.2 POSIT COMO ALTERNATIVA

As limitações observadas nos formatos tradicionais de baixa precisão, como Float8, bfloat16 e representações fixas, motivam a busca por sistemas numéricos mais adequados às exigências do treinamento de redes neurais profundas. O sistema Posit surge como uma dessas alternativas, especialmente por apresentar propriedades

alinhas às características estatísticas e operacionais dessas redes.

Durante o treinamento, é comum que a maior parte dos valores esteja concentrada em torno de zero. Formatos como o Float8 tendem a anular essas quantidades pequenas, comprometendo a atualização dos parâmetros e, consequentemente, o processo de aprendizado. Já o Posit, ao distribuir os valores representáveis de forma densa na vizinhança do zero, permite que variações pequenas permaneçam representáveis, mesmo em representações com 8 bits.

O formato Posit lida com a necessidade de precisão relativa maior para valores com ordem de grandeza próxima da unidade, onde ocorrem as principais ativações e pesos por meio de sua precisão variável, que ajusta dinamicamente a quantidade de bits da mantissa com base no valor representado. Isso permite que a precisão seja alocada de forma inteligente, sem necessidade de heurísticas adicionais ou escalonamento explícito dos dados.

Adicionalmente, o Posit mantém consistência e simplicidade nas operações aritméticas básicas. Essas propriedades tornam o formato particularmente atrativo como sistema numérico de baixa precisão para aprendizado profundo, abrindo caminho para sua adoção como substituto direto do ponto flutuante tradicional em contextos onde eficiência computacional e economia de memória são prioritárias.

## 4 APLICAÇÃO DE POSIT EM PINNS

A flexibilidade do sistema Posit e a generalidade da biblioteca PositPR.jl abrem caminho para sua aplicação em modelos numéricos baseados em aprendizado de máquina. Este capítulo descreve como foi realizada a adaptação da infraestrutura computacional para possibilitar o uso de tipos Posit em bibliotecas como Lux.jl e NeuralPDE.jl, incluindo os desafios da inicialização de parâmetros e a geração de números aleatórios com distribuição controlada.

### 4.1 PINNS: REDES NEURAIS INFORMADAS PELA FÍSICA

Redes Neurais Informadas pela Física (PINNs) são uma abordagem moderna na área de Aprendizado de Máquina Científico (SciML, do inglês *Scientific Machine Learning*), voltada para a solução de Equações Diferenciais Parciais (EDPs) e problemas inversos relacionados a modelos físicos. A proposta central das PINNs é incorporar o conhecimento das leis físicas diretamente na função de perda da rede neural, permitindo que o aprendizado ocorra não apenas a partir de dados, mas também a partir da estrutura matemática subjacente ao sistema estudado.

Essa abordagem resolve um dos principais desafios das técnicas clássicas, como elementos finitos ou métodos espectrais: a dificuldade em integrar dados experimentais ou ruidosos em modelos complexos. Além disso, PINNs não exigem malhas regulares nem discretizações explícitas do domínio, o que facilita a aplicação em geometrias complicadas ou problemas mal-postos.

### 4.2 ADAPTAÇÃO DE BIBLIOTECAS

Na linguagem Julia, a implementação de PINNs é facilitada pela biblioteca NeuralPDE.jl (Zubov et al., 2021), desenvolvida como parte do ecossistema SciML. Essa biblioteca permite a formulação de problemas envolvendo EDPs de forma declarativa, em que o usuário especifica os termos da equação diferencial, as condições de contorno e/ou iniciais e o domínio do problema. A partir dessa especificação simbólica, o pacote constrói automaticamente a função de perda física e fornece mecanismos de treinamento compatíveis com fluxos automáticos de diferenciação.

A biblioteca NeuralPDE.jl utiliza a biblioteca Lux.jl (Pal, 2023a) como *backend* para definição das arquiteturas neurais, gerenciamento de parâmetros e suporte à diferenciação automática. Essa separação modular permite que diferentes arquiteturas sejam experimentadas com facilidade, incluindo redes profundas, convolucionais ou ar-

quiteturas personalizadas, desde que implementadas em conformidade com a interface de Lux.

Com a biblioteca PositPR.jl funcionando de forma independente, foi possível iniciar sua integração com as bibliotecas de aprendizado de máquina baseadas em Julia (NeuralPDE.jl e Lux.jl). Para que fosse possível utilizar redes parametrizadas com o tipo Posit, foi necessário modificar a forma como a biblioteca Lux.jl reconhece e constrói tensores numéricos com valores iniciais. Isso envolveu a adaptação de funções auxiliares de inicialização, como zeros, ones, rand e randn, para que aceitassem os tipos definidos na PositPR.jl, como Posit{8,1}, Posit{8,2}, entre outros. Nesse contexto, como o tipo Posit herda da abstração Real, as chamadas nativas a rand e randn feitas pela Lux.jl passam a reconhecer as versões especializadas implementadas em PositPR.jl, garantindo que toda a cadeia de inicialização funcione de forma transparente e consistente com o novo sistema numérico.

Ressalta-se que o funcionamento detalhado de rand e randn, bem como sua importância para inicializar redes neurais com Posit, será explicado na Apartado 4.3, enquanto exemplos práticos de aplicação serão apresentados na Apartado 5.2.

Essas alterações permitiram que as funções de inicialização da Lux.jl, como randnP16\_2 ou onesP8\_2, funcionassem normalmente com tipos Posit. Com isso, toda a cadeia de chamadas internas – desde a criação dos pesos e vieses até o uso dessas estruturas no treinamento – passou a aceitar a nova representação numérica.

Essas mudanças são fundamentais para que os modelos construídos em NeuralPDE.jl possam ser executados sem depender da conversão para ponto flutuante padrão. Isso permite investigar, com maior controle, os efeitos da representação numérica nas aproximações.

Uma das principais dependências dessa adaptação é a correta geração dos valores iniciais. Como muitas redes neurais são sensíveis à forma como os parâmetros são inicializados, é essencial que a geração de números aleatórios em Posit preserve propriedades como uniformidade ou distribuição normal.

Vale destacar que a biblioteca Lux.jl, embora utilizada neste projeto como *backend* da NeuralPDE.jl, também serve como infraestrutura de redes neurais para outras bibliotecas do ecossistema Julia. Exemplos notáveis incluem DiffEqFlux.jl (Rackauckas et al., 2020), que combina aprendizado profundo com resolução de EDOs, e GalacticOptim.jl (Dixit; Rackauckas, 2023), voltada para otimização baseada em gradientes ou métodos evolutivos. Assim, adaptar o suporte ao tipo Posit na Lux.jl abre possibilidades para uma gama mais ampla de aplicações em SciML, indo além das PINNs.

## 4.3 GERADOR DE NÚMEROS ALEATÓRIOS EM POSIT

Uma etapa crítica na construção de modelos baseados em redes neurais é a geração dos valores iniciais para pesos e vieses. Para que experimentos utilizando o sistema Posit sejam válidos e reproduzam o comportamento estatístico esperado, é necessário garantir que os números aleatórios gerados sigam distribuições compatíveis com o tipo de inicialização tradicionalmente usado em ponto flutuante, como a distribuição uniforme em  $[0, 1]$  ou a normal padrão.

### 4.3.1 Distribuição Uniforme

Para gerar números aleatórios uniformemente distribuídos no intervalo  $[0, 1]$  com precisão controlada por  $\text{Posit}\{ps, es\}$ , foi utilizada uma técnica que evita a conversão direta de ponto flutuante. Essa abordagem consiste em fixar o regime e o expoente, restringindo a geração dos números ao intervalo  $[1/2, 1)$ . Dentro deste intervalo, é possível aplicar uma transformação linear da forma  $2x - 1$  para obter uma distribuição uniforme em  $[0, 1]$  (Algoritmo 1).

A motivação para essa técnica está no fato de que as operações de multiplicação por 2 e subtração de 1 preservam a estrutura uniforme dos valores de entrada, sobretudo quando o intervalo de entrada está contido em uma região onde a malha do Posit é densa e bem distribuída, como ocorre com  $[1/2, 1)$  e  $[1, 2)$ .

---

#### Algoritmo 1 Geração de número aleatório uniforme em $[0, 1]$ com Posit

---

```

1: function POSIT_RAND(ps, es)
2:   regime  $\leftarrow$  "01"                                 $\triangleright$  regime fixado para  $k = -1$ 
3:   e_bits  $\leftarrow$  "1" repetido es vezes           $\triangleright$  expoente máximo
4:   f_bits  $\leftarrow$  ps  $- (1 + \text{length}(\text{regime}) + es)       $\triangleright$  bits restantes para fração
5:   fração  $\leftarrow$  RAND_BITS_STRING(f_bits)            $\triangleright$  bits aleatórios
6:   b  $\leftarrow$  "0"  $\parallel$  regime  $\parallel$  e_bits  $\parallel$  fração       $\triangleright$  concatenação dos campos
7:   p  $\leftarrow$  POSITDECODER(b, ps, es)                   $\triangleright$  transforma  $[1/2, 1)$  em  $[0, 1)$ 
8:   return  $2 \cdot p - 1$ 
9: end function$ 
```

---

Entretanto, essa abordagem apresenta limitações quando utilizada com números Posit de 8 bits ( $\text{Posit}\{8, es\}$ ), isto é, para valores de precisão reduzida. Nesses casos, o domínio atingido pelas operações aritméticas é restrito a regimes com  $k \geq -1$ , o que impede a geração de valores pequenos, próximos de zero. Isso prejudica a alta densidade de representações próximas da origem. Para contornar esse problema, optou-se por uma abordagem híbrida: especificamente para geração de números aleatórios em Posit de 8 bits, é utilizado o formato Float64 para gerar números reais uniformemente distribuídos em  $[0, 1]$  que, em seguida, são convertidos para Posit com precisão limitada. Apesar de se tratar de uma conversão – algo evitado neste trabalho –,

esse procedimento pontual é justificado pelo fato de que, nesse cenário, a distribuição estatística desejada é preservada sem comprometer significativamente a integridade da representação Posit, além de ser aplicada apenas na etapa de geração dos dados iniciais.

Para valores de  $ps$  maiores, essa restrição não é necessária, pois a maior resolução oferecida permite que a técnica baseada na manipulação de bits em  $[1/2, 1)$  funcione adequadamente. Assim, a escolha da estratégia de geração aleatória depende diretamente do tipo Posit utilizado e de sua capacidade de representar valores próximos de zero com boa granularidade.

#### 4.3.2 Distribuição Normal

Em diversos métodos de inicialização de redes neurais, distribuições normais são preferidas em relação a distribuições uniformes, principalmente por suas propriedades estatísticas centradas em torno da média zero. Para gerar números aleatórios com distribuição normal padrão, utilizamos a transformação de Box-Muller (Box; Muller, 1958), que permite gerar duas variáveis com distribuição normal padrão a partir de duas variáveis uniformemente distribuídas no intervalo  $(0, 1)$ .

A transformação utilizada é expressa pelas fórmulas:

$$z_0 = \sqrt{-2 \ln u_1} \cdot \cos(2\pi u_2), \quad z_1 = \sqrt{-2 \ln u_1} \cdot \sin(2\pi u_2),$$

em que  $u_1$  e  $u_2$  são variáveis com distribuição uniforme em  $(0, 1)$  e  $z_0$  e  $z_1$  possuem distribuição normal padrão. Toda a transformação foi implementada diretamente na aritmética Posit, utilizando a biblioteca PositPR.jl.

A obtenção das variáveis uniformes  $u_1$  e  $u_2$  envolve duas etapas: primeiro, a criação de valores uniformemente distribuídos (por geração direta ou conversão, a depender do parâmetro  $ps$ ) e, em seguida, a aplicação da transformação de Box-Muller. Essa abordagem torna possível inicializar redes neurais com pesos normalmente distribuídos.

## 5 ANÁLISE EXPERIMENTAL E RESULTADOS

Este capítulo apresenta a investigação empírica sobre o comportamento do sistema numérico Posit, com foco na avaliação de sua precisão, densidade de representação, estabilidade das operações aritméticas e seu desempenho na inicialização e no comportamento no treinamento de redes neurais.

### 5.1 AVALIAÇÃO DE PRECISÃO NUMÉRICA DE POSITPR.JL

#### 5.1.1 Precisão e Densidade de Representações Numéricas

Para avaliar a precisão intrínseca das representações Posit, foi calculado o erro absoluto entre o valor real  $x$  e sua representação reconstruída  $\tilde{x}$  após codificação e decodificação.

Seja  $x$  um número representável em `Float64`. Denote por  $\tilde{x}$  sua aproximação em um tipo numérico  $T$ , isto é,

$$\tilde{x} := \text{decode}(\text{encode}(x, T)).$$

No contexto de `PositPR.jl`, essa aproximação é dada pela função `x2Posit`, que produz um número em  $T = \text{Posit}\{\text{ps}, \text{es}\}$ . É interessante observar que Posits de baixa precisão estão contidos em `Float64`.

As Figuras 3a y 3b mostram o erro absoluto  $|x - \tilde{x}|$  ao longo do domínio positivo para duas configurações distintas de Posit. Observa-se que, quanto maior o número total de bits (`ps`), menor o erro de representação e maior a densidade de pontos próximos da unidade. As regiões com espaçamentos abruptos correspondem às transições entre regimes, evidenciando a perda de resolução local.

É importante destacar também a ordem de grandeza dos erros: em configurações de menor precisão, como no caso de 8 bits, surgem lacunas significativas na representação numérica. Entre os valores 16 e 20, por exemplo, não há nenhum número representável, o que faz com que a magnitude do erro se eleve de forma acentuada. Esse comportamento ressalta como a capacidade de representação do Posit se distribui de maneira não uniforme, privilegiando regiões próximas a 0.

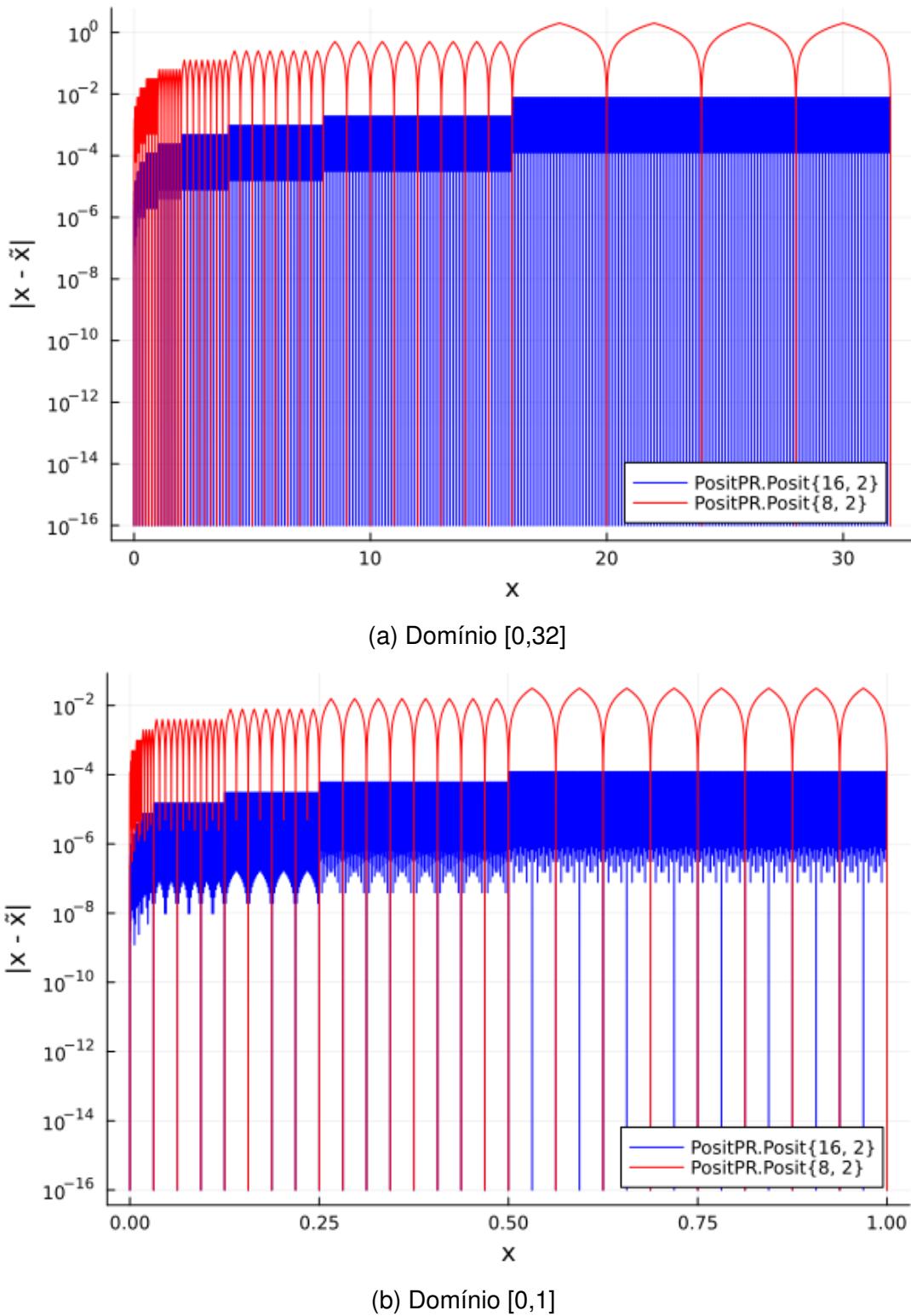


FIGURA 3 – Erro de representação para Posit{16, 2} e Posit{8, 2}, em escala logarítmica.

Uma das características mais relevantes do sistema Posit é sua capacidade de representar números com maior densidade nas proximidades da origem. Na figura 4, foi calculado o espaçamento absoluto entre dois valores consecutivos representáveis no intervalo  $(0, 1)$ , utilizando três configurações de Posit: Posit{8,1}, Posit{8,2} e

`Posit{16,2}`. O espaçamento local é definido como

$$\Delta x_i = x_{i+1} - x_i,$$

onde  $x_i$  e  $x_{i+1}$  são dois valores consecutivos representáveis por Posit em ordem crescente, ou seja,  $x_{i+1} = \text{nextPosit}(x_i)$ . Os valores de  $\Delta x_i$  foram plotados em escala logarítmica para ressaltar a ordem de magnitude da densidade numérica em cada região. Observa-se, neste caso, a ordem de magnitude do menor incremento capaz de modificar a representação numérica (`eps`) associado a zero para os três formatos, destacados em linhas verticais na Figura 4.

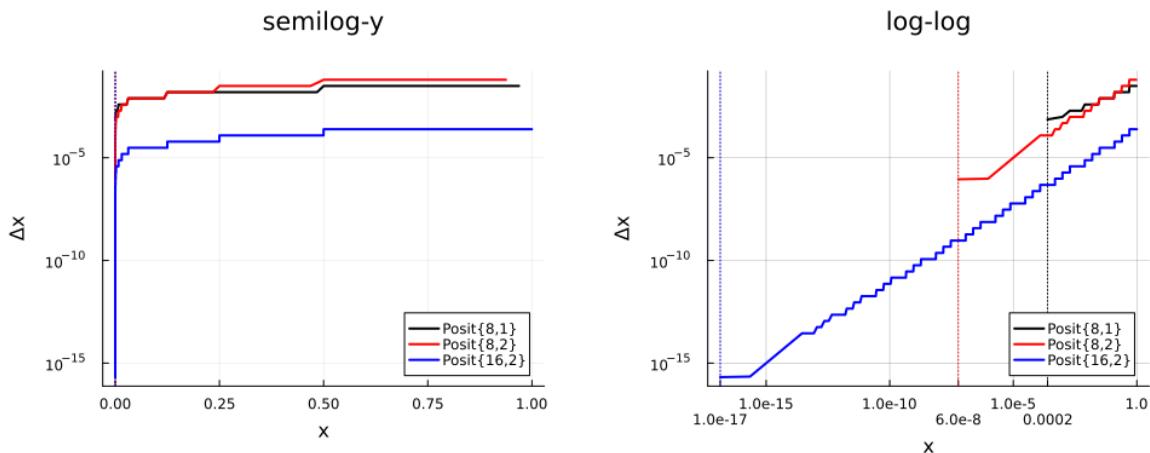


FIGURA 4 –  $\Delta x$  entre valores representáveis em  $(0, 1]$  para diferentes configurações de Posit, em escala logarítmica.

### 5.1.2 Operações Aritméticas Elementares

Para investigar o comportamento das operações aritméticas, definimos um subespaço do domínio  $\mathbb{R}^2$  a partir de pares  $(x, y)$  em uma malha regular e aplicamos a conversão explícita de cada valor real para sua representação Posit antes da operação.

Sejam  $x, y \in \mathbb{R}$ , e seja  $T$  uma configuração Posit específica (como `Posit{8,2}`, por exemplo). Aplicamos a transformação

$$\tilde{x} := \text{x2Posit}(x, T), \quad \tilde{y} := \text{x2Posit}(y, T)$$

e, em seguida, avaliamos a operação binária desejada  $f(\cdot, \cdot)$  no espaço Posit. O resultado é reconvertido para ponto flutuante padrão com

$$z := \text{Posit2x}(f(\tilde{x}, \tilde{y})).$$

Esse procedimento permite observar, em cada ponto da malha  $(x, y)$ , como as operações aritméticas elementares: soma (Figura 5), subtração (Figura 6), multiplicação (Figura 7), divisão (Figuras 8 y 9) e potenciação (Figura 10) se comportam quando executadas com operandos representáveis no sistema Posit. A visualização resultante em

forma de mapa de calor evidencia características como regiões de saturação, descontinuidades numéricas, simetria (ou falta dela) e resolução variável, todas dependentes da configuração de bits e expoentes utilizados.

Nota-se, na Figura 10, a presença de uma faixa onde a operação  $\sqrt{x}$  é utilizada em substituição direta à potenciação  $x^{1/2}$ . Essa escolha se justifica pelo fato de que a função raiz quadrada apresenta um comportamento numérico mais estável e preciso em representações de baixa precisão, como é o caso dos formatos Posit{8,1} e Posit{8,2}.

+ PositPR.Posit{8, 1}      + PositPR.Posit{8, 2}      + PositPR.Posit{16, 2}

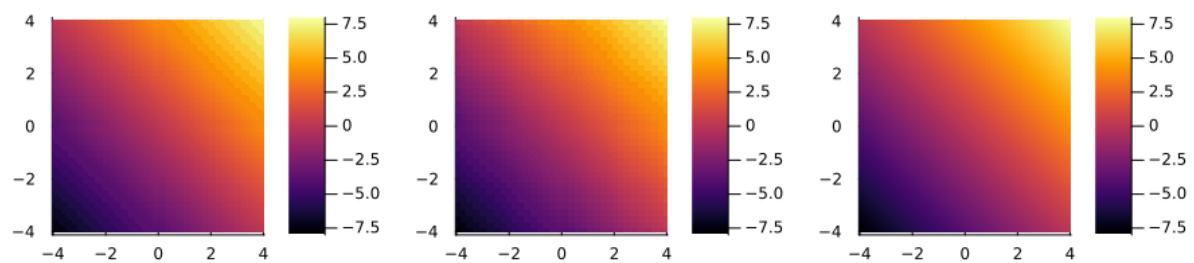


FIGURA 5 –  $x + y$  para diferentes configurações Posit.

- PositPR.Posit{8, 1}      - PositPR.Posit{8, 2}      - PositPR.Posit{16, 2}

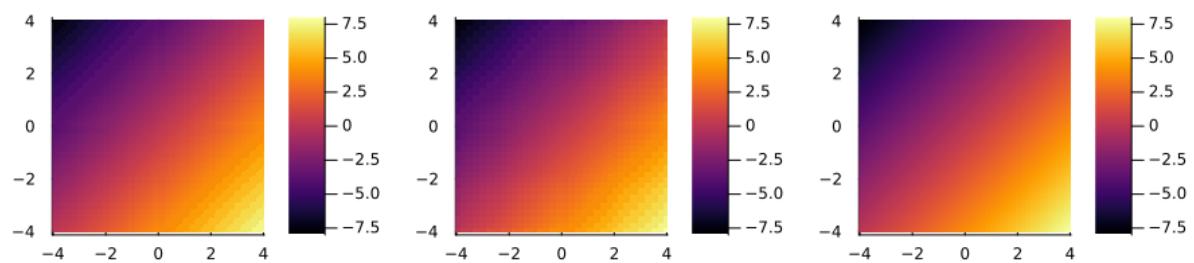


FIGURA 6 –  $x - y$  para diferentes configurações Posit.

\* PositPR.Posit{8, 1}      \* PositPR.Posit{8, 2}      \* PositPR.Posit{16, 2}

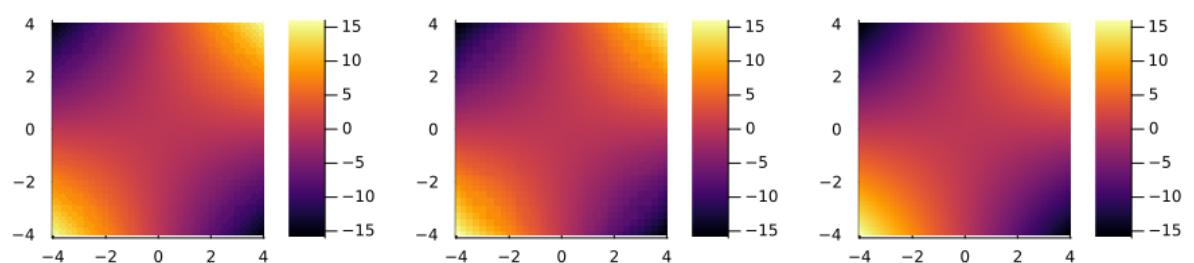


FIGURA 7 –  $x \cdot y$  para diferentes configurações Posit.

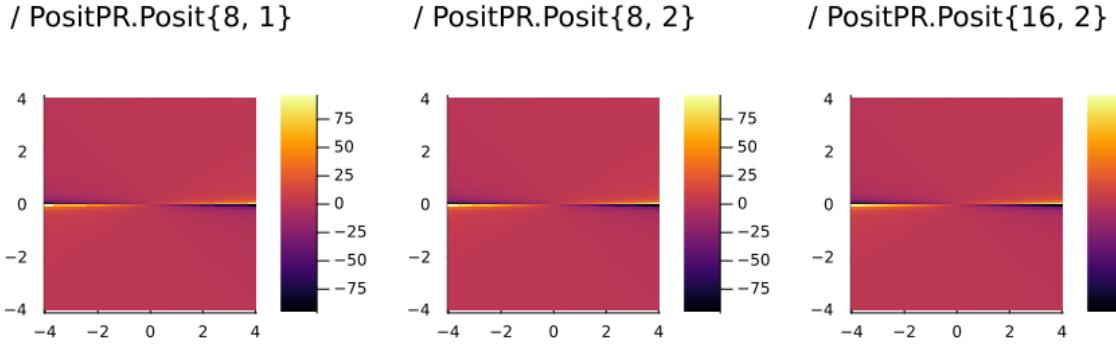


FIGURA 8 –  $x/y$  para diferentes configurações Posit.

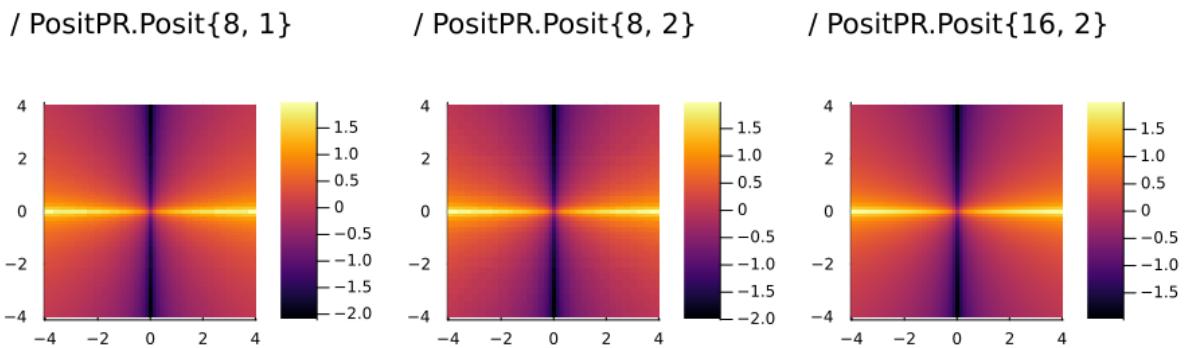


FIGURA 9 –  $\log(|x/y|)$  para diferentes configurações Posit.

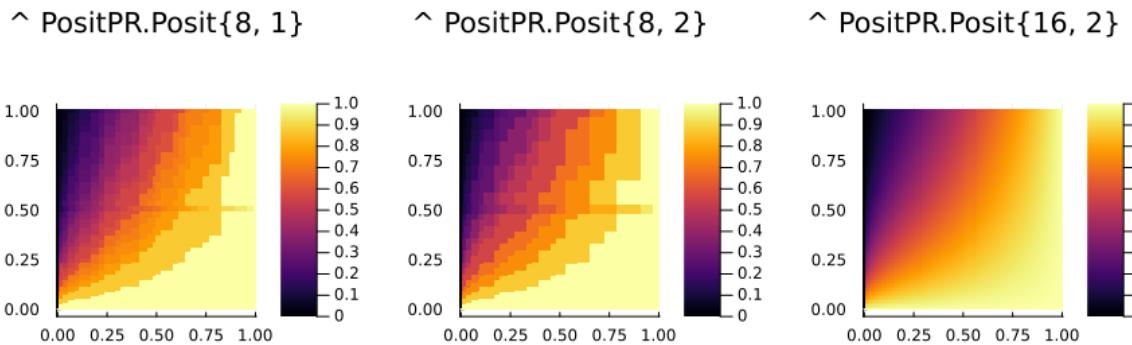


FIGURA 10 –  $x^y$  para diferentes configurações Posit (com  $x, y \in [0, 1]$ ).

### 5.1.3 Avaliação de Funções

A maioria das funções avaliadas nesta seção está descrita na Tabela 1, apresentada no Capítulo 2. As figuras a seguir ilustram essas funções sob diferentes configurações Posit{ps,es}, algumas com o objetivo de demonstrar o comportamento geral da representação Posit, e outras com foco na análise comparativa de erros numéricos.

A Figura 11 ilustra a avaliação de  $f(x) = \log(x)$ ,  $f(x) = \sqrt{x}$  e  $f(x) = x^2$ . Na Figura 12, é apresentada uma comparação entre duas abordagens para a função identidade  $f(x) = x$ . A Figura 14 exibe o comportamento de funções com crescimento exponencial. Já a Figura 15 contempla as funções trigonométricas elementares,  $f(x) = \sin(x)$  e  $f(x) = \cos(x)$ . Singularidades e regiões mal-condicionadas, como  $f(x) = 1/x$  e  $f(x) = \frac{x}{x+1}$ , são analisadas na Figura 16, por sua relevância em testar a estabilidade numérica. Por fim, a Figura 17 aborda funções comumente utilizadas como funções de ativação não lineares no contexto de redes neurais, tais como  $f(x) = \max(0, x)$  (ReLU) e  $f(x) = \tanh(x)$ , destacando seu papel na saturação e na propagação de gradientes durante o treinamento.

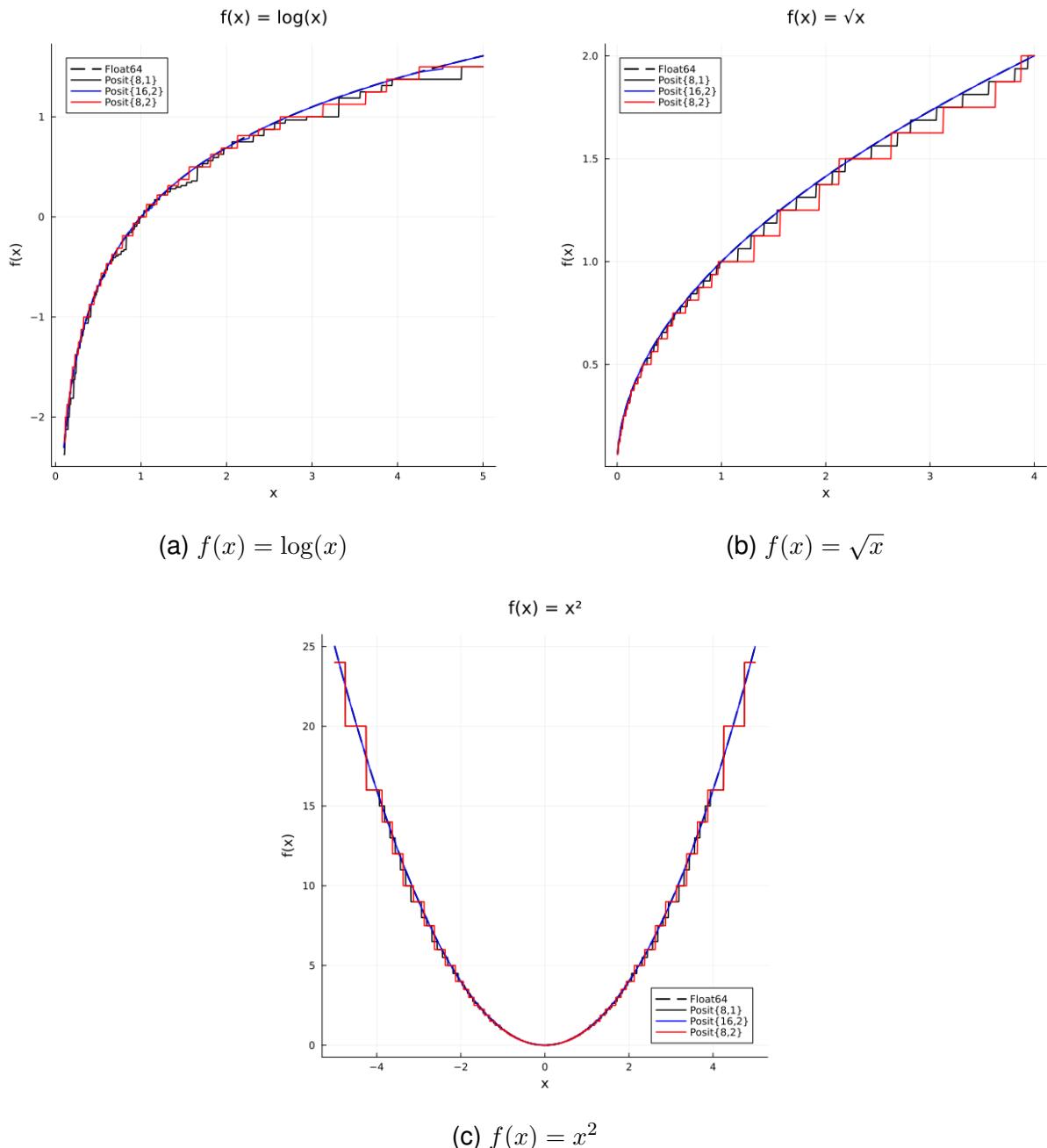
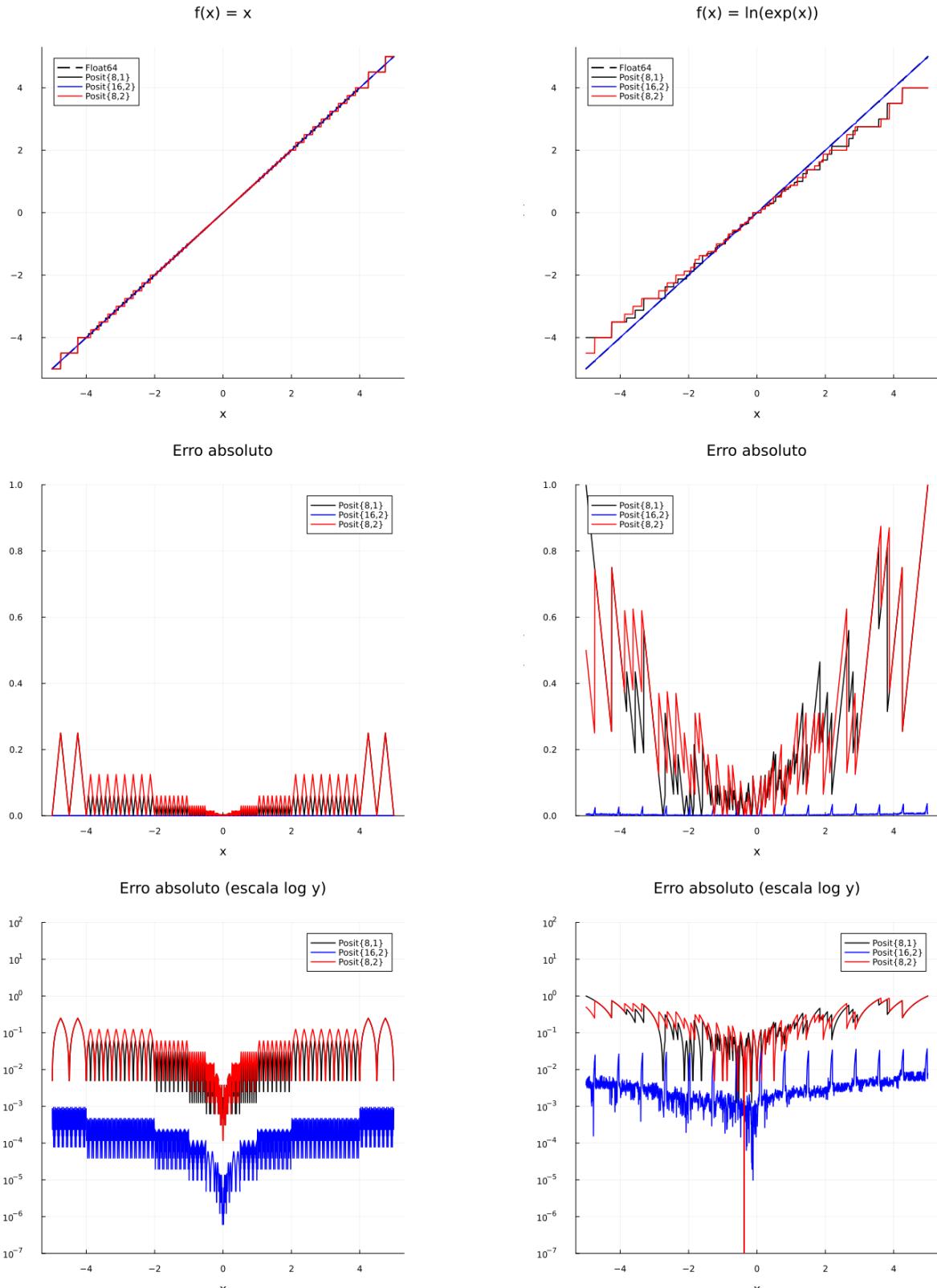


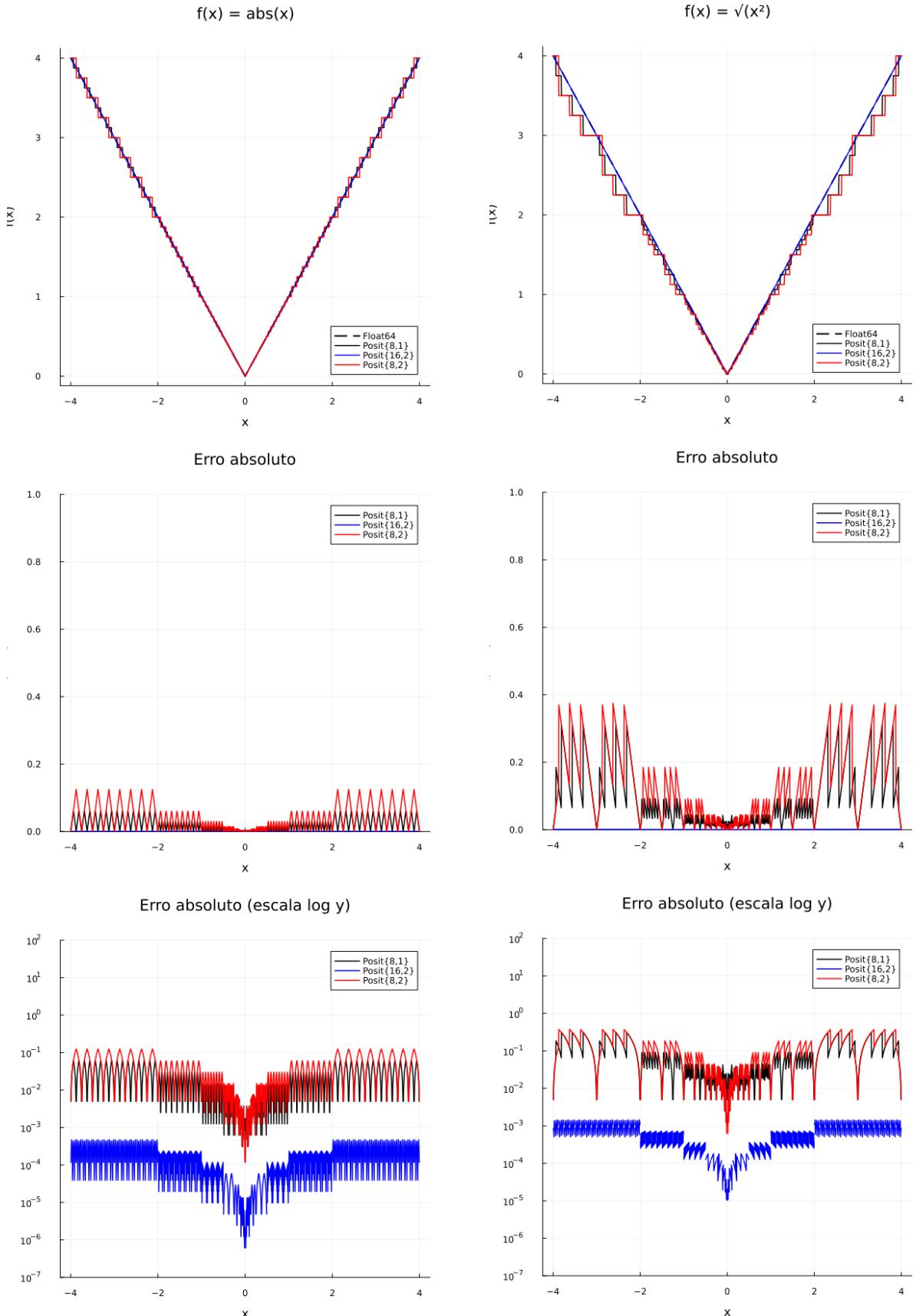
FIGURA 11 – Avaliação das funções  $\log(x)$ ,  $\sqrt{x}$  e  $x^2$  utilizando diferentes representações Posit.



(a) Avaliação de  $f(x) = x$  com diferentes configurações Posit.

(b) Avaliação de  $f(x) = \ln(e^x)$  com diferentes configurações Posit.

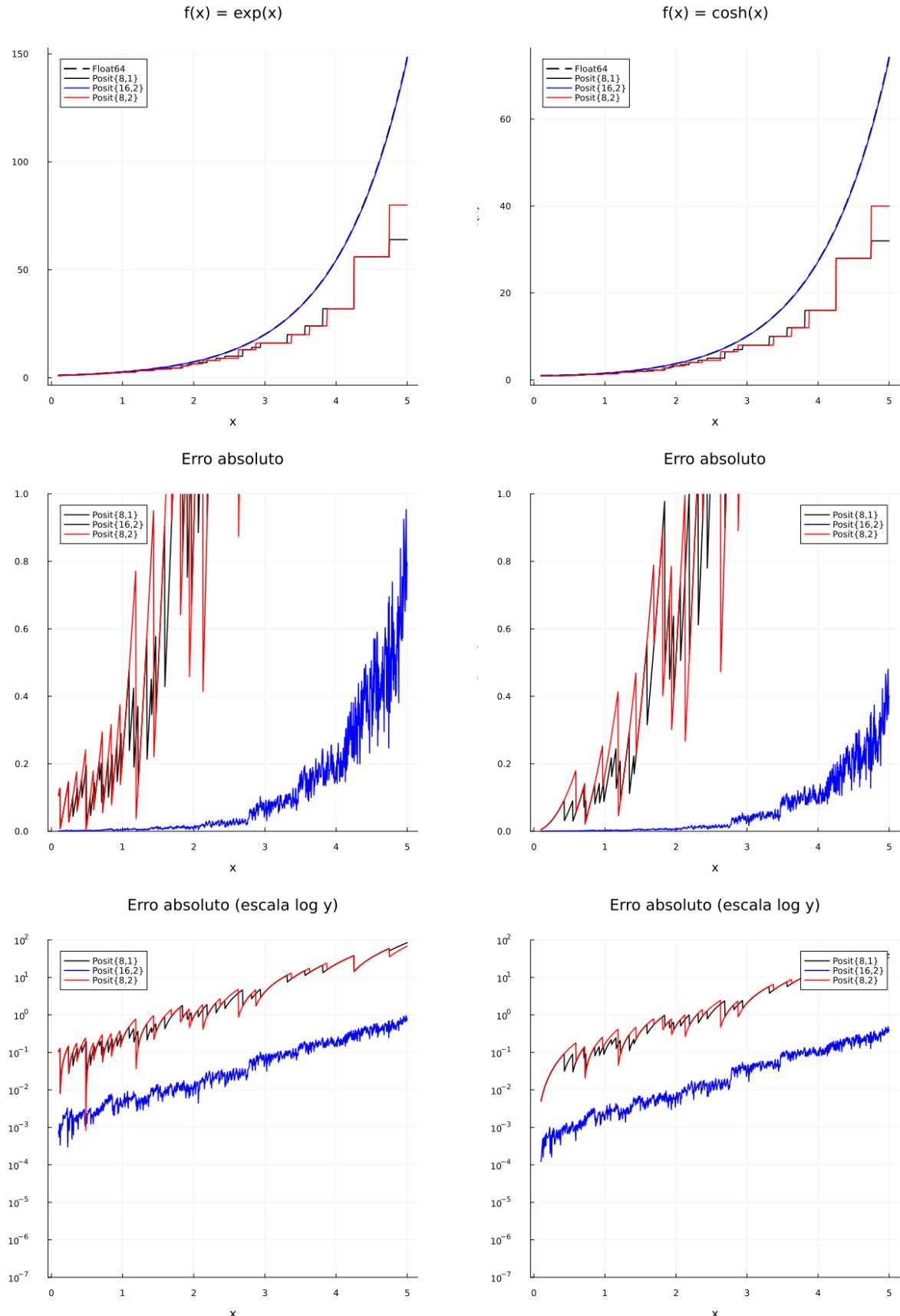
FIGURA 12 – Comparação entre  $f(x) = x$  e  $f(x) = \ln(e^x)$  utilizando diferentes representações Posit.



(a) Avaliação de  $f(x) = |x|$  com diferentes configurações Posit.

(b) Avaliação de  $f(x) = \sqrt{x^2}$  com diferentes configurações Posit.

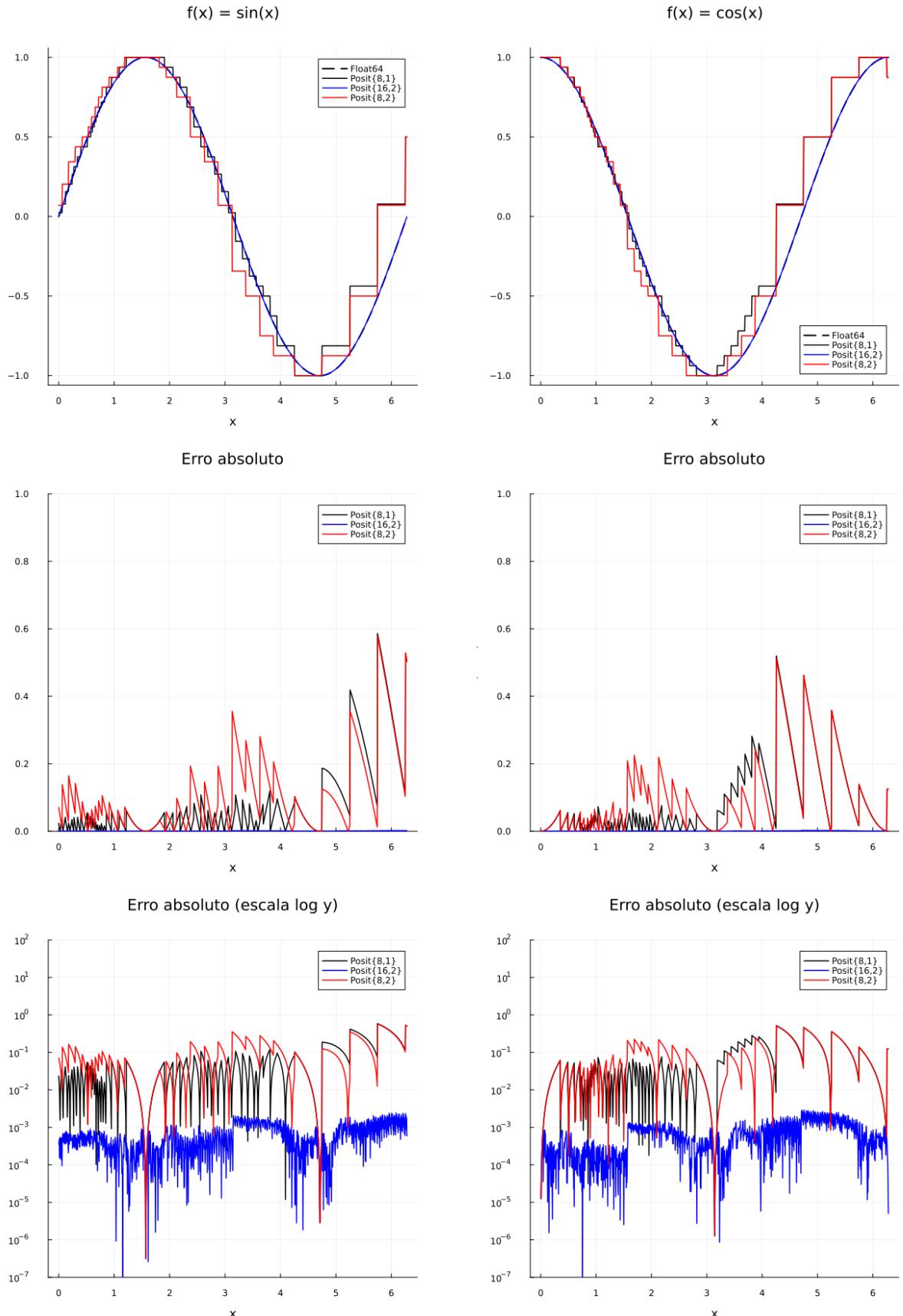
FIGURA 13 – Comparação entre  $f(x) = |x|$  e  $f(x) = \sqrt{x^2}$  utilizando diferentes representações Posit.



(a) Avaliação de  $f(x) = e^x$  com diferentes configurações Posit.

(b) Avaliação de  $f(x) = \cosh(x)$  com diferentes configurações Posit.

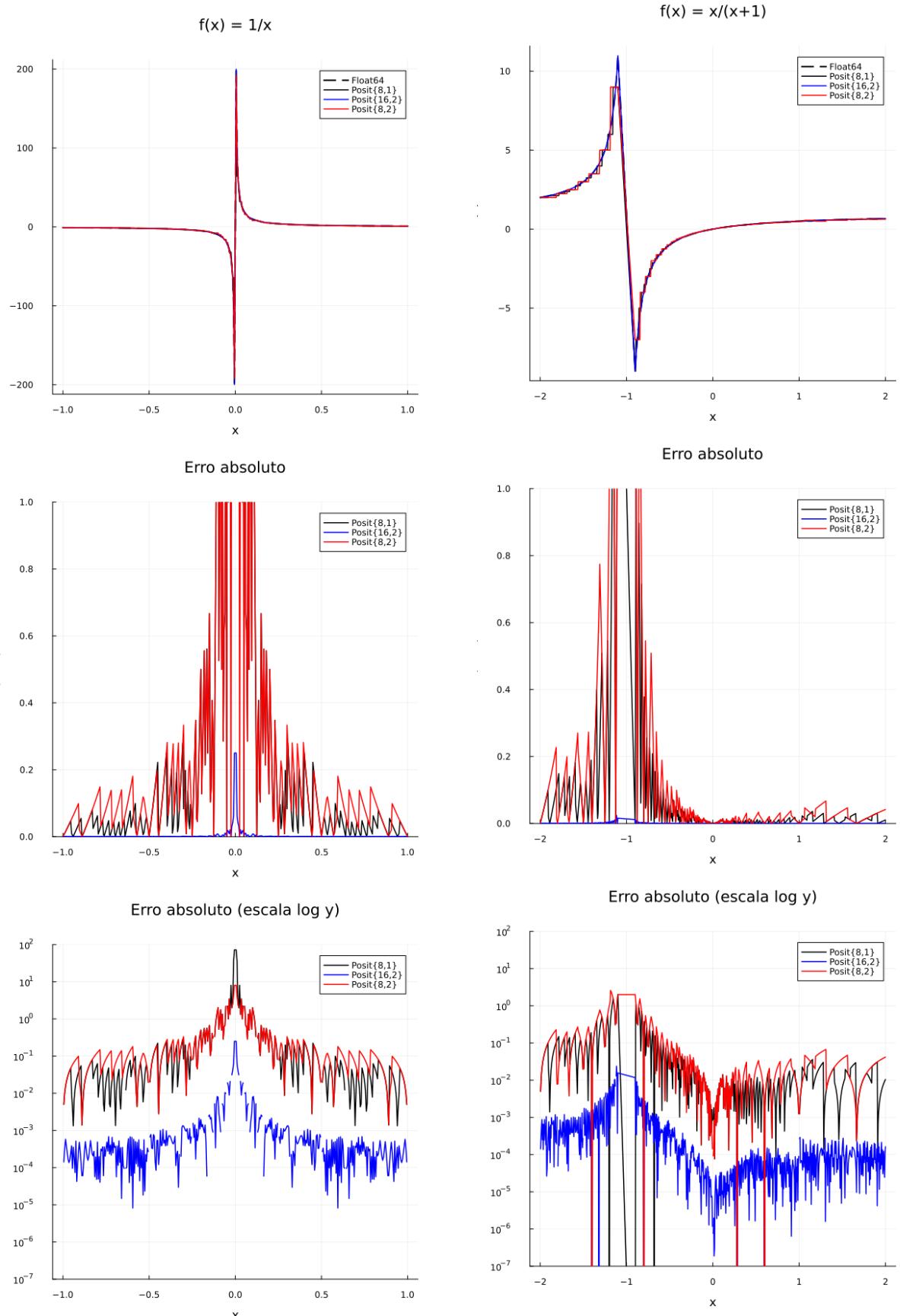
FIGURA 14 – Comparação entre  $f(x) = e^x$  e  $f(x) = \cosh(x)$  utilizando diferentes representações Posit.



(a) Avaliação de  $f(x) = \sin(x)$  com diferentes configurações Posit.

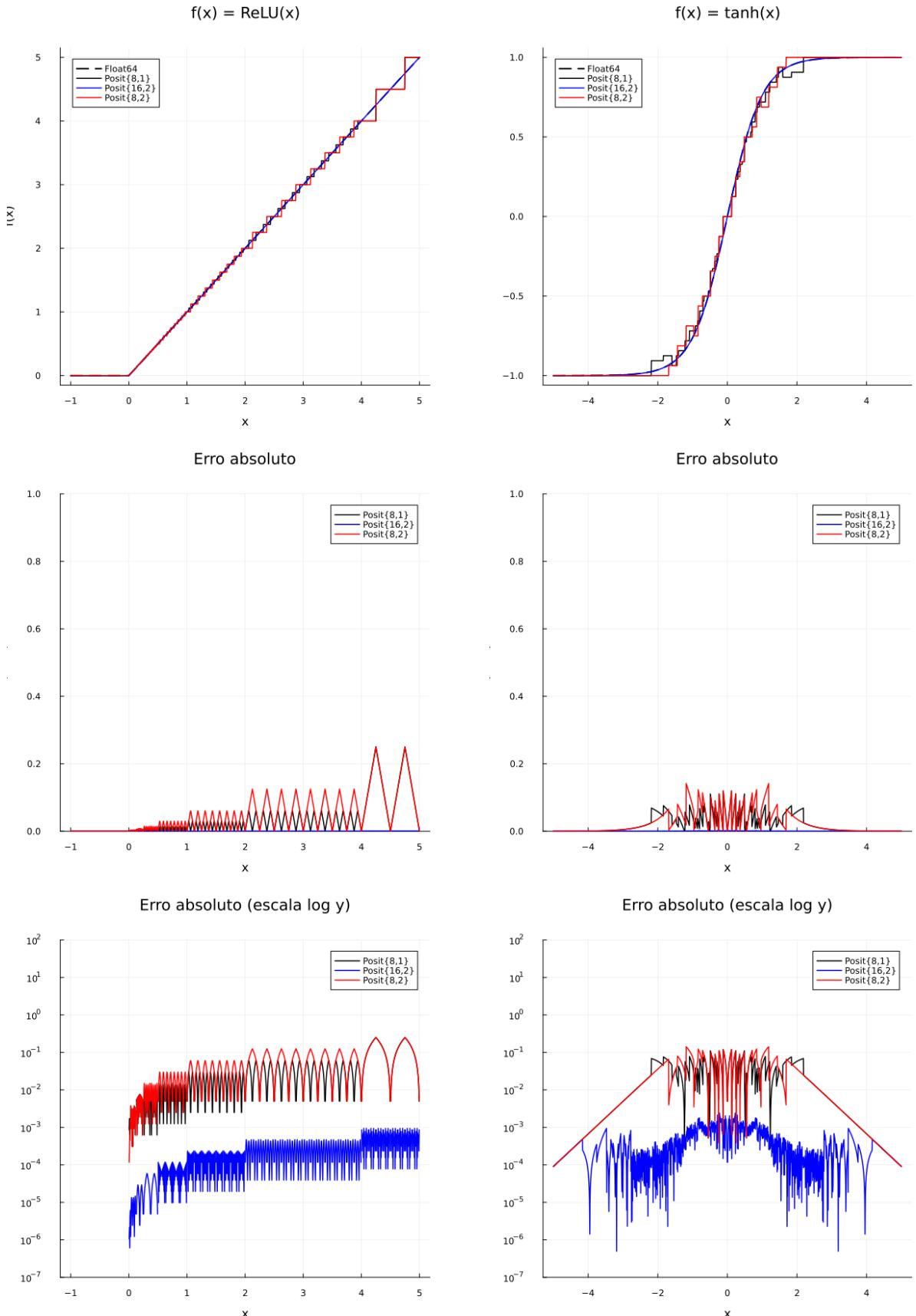
(b) Avaliação de  $f(x) = \cos(x)$  com diferentes configurações Posit.

FIGURA 15 –  $f(x) = \sin(x)$  e  $f(x) = \cos(x)$  utilizando diferentes representações Posit.



(a)  $f(x) = 1/x$  no domínio  $[-1, -\Delta x] \cup [\Delta x, 1]$ ; o polo em  $x = 0$  foi excluído. (b)  $f(x) = \frac{x}{x+1}$  no domínio  $[-2, -(\Delta x+1)] \cup [(\Delta x-1), 2]$ ; o polo em  $x = -1$  foi excluído.

FIGURA 16 – Comparação entre  $f(x) = 1/x$  e  $f(x) = x/(x+1)$  utilizando diferentes representações Posit.



(a) Avaliação de  $f(x) = \text{ReLU}(x)$  com diferentes configurações Posit.

(b) Avaliação de  $f(x) = \tanh(x)$  com diferentes configurações Posit.

FIGURA 17 – Comparação entre  $f(x) = \text{ReLU}(x)$  e  $f(x) = \tanh(x)$  utilizando diferentes representações Posit.

## 5.2 APLICAÇÕES EM REDES NEURAIS

### 5.2.1 Inicialização de Redes (Lux.jl)

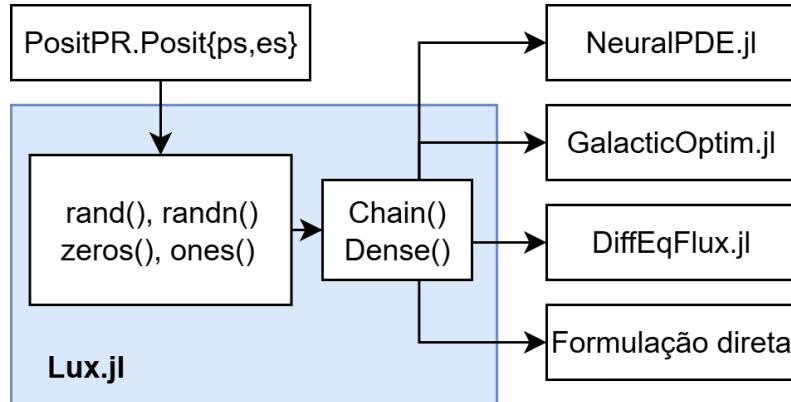


FIGURA 18 – Integração da biblioteca PositPR.jl com o ecossistema SciML.

A Figura 18 ilustra o papel da biblioteca PositPR.jl como núcleo da representação numérica baseada em Posits, conectando-se diretamente à biblioteca Lux.jl, que por sua vez serve como base para frameworks de aprendizado de máquina no ecossistema Julia, como NeuralPDE.jl, GalacticOptim.jl, DiffEqFlux.jl e implementações manuais. Essa integração viabiliza a construção de redes neurais inteiramente operando em aritmética Posit.

No contexto da biblioteca Lux.jl, é possível utilizar diretamente as funções de inicialização com Posit na construção das camadas da rede. Para isso, foram adaptadas as funções `randP{ps}_es`, `randnP{ps}_es`, `zerosP{ps}_es` e `onesP{ps}_es`, de modo que retornem tensores Posit compatíveis com as estruturas esperadas pela biblioteca Lux.jl, como discutido no Capítulo 4.

Para exemplificar, considere a construção de uma rede neural composta por uma camada densa, inicializada com valores em `Posit{8,2}`. A definição da rede pode ser feita com a seguinte estrutura:

```
julia> chain = Chain(
    Dense(1,12,init_weight=randP8_2,init_bias=zerosP8_2),
    Dense(12, 1,init_weight=randP8_2,init_bias=zerosP8_2)
)

rng = Random.default_rng()
theta,st = Lux.setup(rng,chain)
```

Nesse código, `theta` armazena os parâmetros  $\theta$  treináveis da rede (pesos e vieses), enquanto `st` contém os estados associados às camadas.

Esse modelo é composto por duas camadas sequenciais. Cada uma é construída com inicialização explícita baseada em Posit, tanto para os pesos (`init_weight`) quanto para os vieses (`init_bias`).

A extração direta dos pesos de um neurônio confirma a consistência da representação. Por exemplo, ao acessar os pesos associados ao neurônio 5 da primeira camada densa:

```
julia> ps[:layer_1][:weight][5,:]
1-element Vector{PositPR.Posit{8,2}}:
PositPR.Posit{8,2}(0,0,-1,2,2,3,"110",3)
```

Observa-se que o valor retornado é um objeto `Posit{8,2}` completo, com todos os campos internos devidamente codificados. Isso evidencia que os neurônios da rede armazenam e operam diretamente em aritmética `Posit`.

### 5.2.2 Treinamento da rede

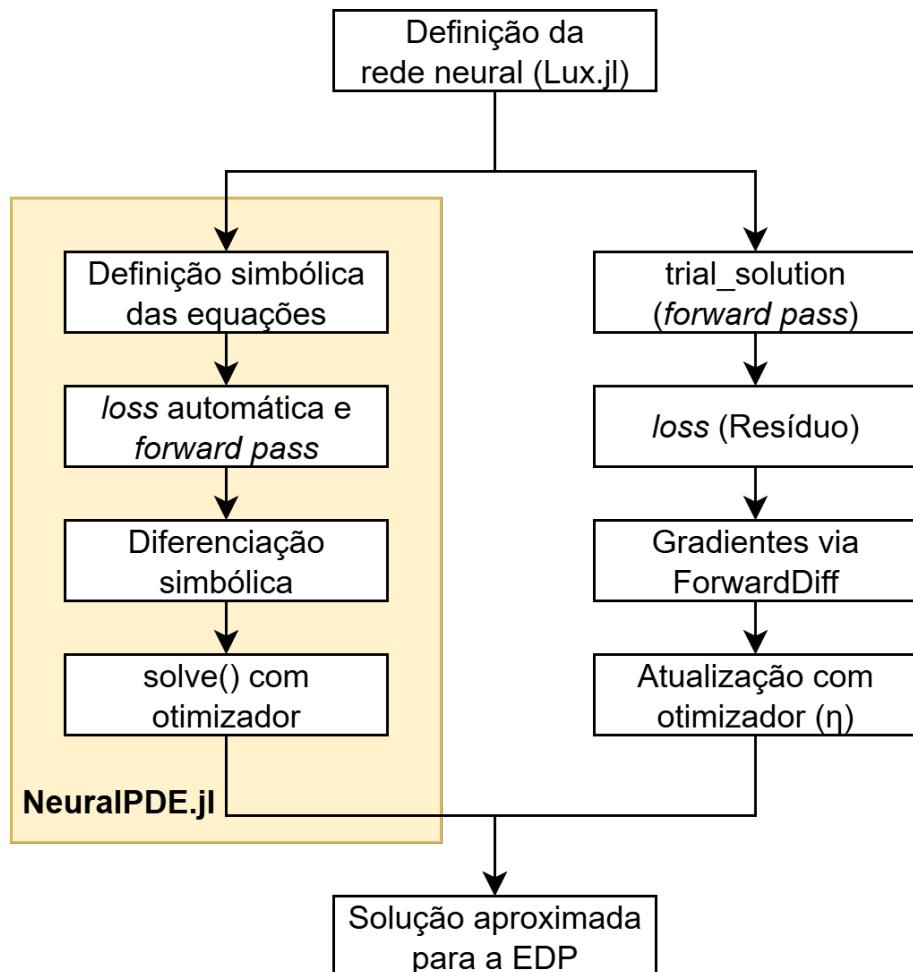


FIGURA 19 – Fluxo de treinamento da rede neural: comparação entre abordagem manual e uso da biblioteca `NeuralPDE.jl`.

A figura 19 compara as duas abordagens utilizadas neste trabalho para o treinamento das PINNs: a implementação manual com controle explícito de cada etapa, e a abstração automática fornecida pela biblioteca NeuralPDE.jl ([código disponível no GitHub](#)). Em ambas, a rede é construída com Lux.jl, mas os mecanismos de propagação direta, cálculo da perda e atualização dos parâmetros diferem substancialmente.

O fluxo de cálculo no treinamento de uma rede neural envolve duas etapas centrais: a propagação direta (*forward pass*) e a retropropagação de gradientes (*backward pass*). A primeira etapa é responsável por gerar a saída da rede a partir de uma entrada, enquanto a segunda calcula os gradientes da função perda e atualiza os pesos e vieses da rede com base nesses valores.

Nesta seção, adotamos duas estratégias distintas para conduzir esse processo. A primeira consiste em uma implementação manual, em que ambas as etapas são construídas diretamente, utilizando Lux.jl e diferenciação automática via ForwardDiff.jl. Já a segunda utiliza a biblioteca NeuralPDE.jl, que oferece uma abstração de alto nível para definição e solução de problemas diferenciais com redes neurais.

A forma como a propagação direta é realizada também depende da abordagem adotada. Na implementação manual, a propagação direta é encapsulada na função `trial_solution`. Por outro lado, quando se utiliza NeuralPDE.jl, essa etapa é internalizada pela estrutura `PhysicsInformedNN`. A retropropagação de gradientes também difere, uma vez que a NeuralPDE.jl utiliza diferenciação simbólica em vez da diferenciação automática direta.

#### 5.2.2.1 Sem NeuralPDE.jl

Tendo definida a rede e seus parâmetros, aplicamos essa estrutura no contexto de uma PINN implementada manualmente, sem o uso do pacote NeuralPDE.jl. Nesse cenário, a função `trial_solution` recebe o vetor de entrada, executa a propagação direta pela rede `chain` – que aplica sequencialmente as camadas densas e suas ativações – e faz a imposição forte de condições iniciais e/ou de contorno.

Por exemplo, para resolver uma EDO com condição inicial homogênea em  $t = 0$ , a solução tentativa é construída de modo que a rede já satisfaça a restrição desde o início, multiplicando a saída pela variável temporal  $t$ . Essa formulação garante que a condição seja atendida para todo o processo de otimização, reduzindo o esforço necessário do algoritmo de treinamento.

Tendo definida a rede e seus parâmetros, aplicamos essa estrutura no contexto de uma PINN implementada manualmente, sem o uso do pacote NeuralPDE.jl. Nesse cenário, a função `trial_solution` recebe o vetor de entrada, executa a propagação direta pela rede `chain` – que aplica sequencialmente as camadas densas e suas

ativações – e faz a imposição forte de condições iniciais e/ou de contorno. Por exemplo, para resolver uma EDO com condição inicial homogênea em  $t = 0$ , multiplicamos o resultado da aplicação da rede pelo termo  $t$ :

---

**Algoritmo 2** Construção de `trial_solution` para uma EDO com condição inicial homogênea em  $t = 0$

---

- 1: Receber entrada  $t$  e parâmetros da rede  $\theta$
  - 2:  $out \leftarrow \text{CHAIN}(t, \theta)$  ▷ propagação direta na rede
  - 3: **return**  $t \cdot out$  ▷ imposição forte da condição inicial
- 

A avaliação da qualidade da solução é feita por meio da função perda (`loss`), que calcula o erro residual associado à EDO ou EDP sendo resolvida. Por exemplo, para uma EDO da forma

$$u'(t) = f(u, t),$$

O resíduo em cada ponto é dado pela diferença entre a derivada da solução tentativa  $u'(t)$  – obtida automaticamente via `ForwardDiff` – e o termo exato  $f(u, t)$ . Em seguida, o quadrado desse resíduo é acumulado ao longo de uma malha de pontos no domínio, e a média desses valores define a função perda.

---

**Algoritmo 3** Cálculo da função perda para uma EDO  $u'(t) = f(u, t)$

---

- 1: Definir malha temporal  $t_1, t_2, \dots, t_N$
  - 2:  $s \leftarrow 0$  ▷ acumulador do erro
  - 3: **for**  $t \in \{t_1, t_2, \dots, t_N\}$  **do**
  - 4:    $u \leftarrow \text{TRIAL\_SOLUTION}(t, \theta)$  ▷  $\theta$  = parâmetros
  - 5:    $u'(t) \leftarrow \text{FORWARDDIFF.DERIVATIVE}(u, t)$
  - 6:    $r(t) \leftarrow u'(t) - f(u, t)$
  - 7:    $s \leftarrow s + r(t)^2$
  - 8: **end for**
  - 9: **return**  $s/N$  ▷ média do erro residual
- 

Para o *backward pass*, utiliza-se o `ForwardDiff` para calcular o gradiente da função perda com respeito aos parâmetros da rede e as atualizações são feitas diretamente sobre os tensores `Posit` por meio de uma implementação explícita de algum otimizador, como mostra o Algoritmo 4.

---

**Algoritmo 4** Retropropagação de Gradientes com `Posit` (caso geral)

---

- 1: Definir hiperparâmetros  $(p_1, p_2, \dots, p_n)$  e parâmetros  $(\theta)$
  - 2: **for**  $epoch = 1, 2, \dots, n\_epochs$  **do**
  - 3:    $grad \leftarrow \text{FORWARDDIFF.GRADIENT}(\text{loss}, \theta)$
  - 4:    $\theta \leftarrow \theta - \text{PASSO\_OTIMIZADOR}(grad, p_1, p_2, \dots, p_n)$
  - 5: **end for**
-

Dessa forma, toda a cadeia de propagação, diferenciação automática e cálculo do resíduo é implementada manualmente, evidenciando a flexibilidade de se construir PINNs diretamente com Lux.jl e o sistema numérico Posit, sem recorrer a bibliotecas como NeuralPDE.jl. Naturalmente, em termos de desempenho de otimização e tempo de processamento, essa abordagem ainda não alcança a eficiência de bibliotecas dedicadas e altamente otimizadas disponíveis para a comunidade de SciML.

### 5.2.2.2 Com NeuralPDE.jl

Considere o problema de valor inicial definido pela EDO:

$$\begin{cases} \frac{du}{dt}(t) + u(t) = \sin(2\pi t), & t \in [0, 1] \\ u(0) = 0 \end{cases} \quad (5.1)$$

cuja solução exata é conhecida e pode ser utilizada para avaliação dos métodos numéricos. Esse tipo de equação representa um caso canônico de dinâmica forçada com decaimento, frequentemente utilizado como benchmark para redes PINNs.

A biblioteca NeuralPDE.jl permite a formulação simbólica desse tipo de problema de forma modular, utilizando o ecossistema do ModelingToolkit.jl. A construção do sistema equivalente à equação (5.1) pode ser feita diretamente em Julia.

```
T = Posit{ps,es}
@parameters t
@variables u(..)
Dt = Differential(t)

eq = Dt(u(t))+u(t)-sin(2*pi*t)^0
bcs = [u(0)^0]
domain = [t in Interval(zero(T), one(T))]
@named pde_system = PDESystem(eq,bcs,domain,[t],[u(t)])
```

A partir da definição simbólica da equação diferencial, o problema é discretizado automaticamente e resolvido com redes neurais definidas usando a estrutura de dados da biblioteca Lux.jl. Os algoritmos de otimização utilizados são provenientes de bibliotecas amplamente reconhecidas na comunidade de aprendizado de máquina em Julia, como Optimisers.jl, Optimization.jl e SciMLBase.jl, garantindo robustez e reproduzibilidade ao experimento ([código disponível](#)).

Diferentemente da abordagem manual descrita anteriormente, na qual os otimizadores foram implementados diretamente sobre os tensores Posit, aqui adotamos a estratégia de utilizar implementações padronizadas e bem testadas dos métodos Adam e SGD (*Stochastic Gradient Descent*), otimizadas para integração com as biblio-

tecas mencionadas. Assim, conseguimos utilizar uma função `solve` para otimizar os parâmetros da rede em relação à função perda.

Esse fluxo permite integrar a formulação simbólica do problema com uma rotina de treinamento eficiente, mantendo o uso de números Posit em toda a cadeia de cálculo, inclusive na avaliação da função perda e no processo de retropropagação.

### 5.2.3 Performance da rede

Tendo definido as estratégias de treinamento com o sistema numérico Posit, esta seção é dedicada à avaliação da performance da rede neural em diferentes cenários.

Para isso, serão utilizadas duas equações. A primeira equação (eq1) é uma EDO simples, com solução suave e periódica. A segunda (eq2) corresponde à equação do calor unidimensional, um caso clássico de EDP parabólica, com evolução temporal e solução analítica conhecida. A Tabela 2 apresenta as equações utilizadas nos experimentos, suas respectivas soluções analíticas e as condições iniciais ou de contorno associadas.

TABELA 2 – Equações diferenciais utilizadas nos experimentos, suas condições e soluções analíticas.

	<b>Equação Diferencial</b>	<b>Condições</b>	<b>Solução Analítica</b>
eq1	$\frac{du}{dt} = \cos(2\pi t)$	$u(0) = 0$	$u(t) = \frac{\sin(2\pi t)}{2\pi}$
eq2	$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$	$u(0, t) = u(1, t) = 0$ $u(x, 0) = \sin(\pi x)$	$u(x, t) = \sin(\pi x)e^{-\pi^2 t}$

Cabe destacar que não foi realizada comparação de tempo de execução entre aritmética em ponto flutuante IEEE 754 e os formatos Posit, uma vez que tal comparação não é significativa neste contexto. Isso se deve ao fato de que processadores modernos possuem unidades aritméticas especializadas para operações com Float, implementadas diretamente em hardware, garantindo execução extremamente eficiente. Em contrapartida, a biblioteca PositPR.jl utilizada neste trabalho apenas emula em software o comportamento da aritmética Posit, tornando sua execução inevitavelmente mais lenta. Assim, o foco da análise recai sobre a qualidade da solução aproximada e o comportamento numérico do treinamento em diferentes precisões, e não sobre a performance computacional.

Antes de avaliarmos a qualidade das soluções aproximadas obtidas pelas redes neurais, é necessário entender qual é o limite de precisão imposto pelo próprio sistema numérico utilizado. Como os formatos de baixa precisão, como Posit{8,2}, apresentam uma discretização menos refinada do domínio dos reais, a melhor aproximação da

solução analítica nesses formatos inevitavelmente se afasta da referência em Float64. Esse afastamento não está relacionado à capacidade da rede, mas sim à própria representação numérica.

Para analisar esse efeito com mais clareza, comparamos duas formas distintas de construir a solução analítica: (i) a reconstrução direta da solução usando exclusivamente operações no tipo Posit desejado, respeitando sua semântica aritmética desde o início; e (ii) a avaliação da solução exata em Float64 seguida de conversão ponto a ponto para o tipo Posit. Ambas as abordagens representam, em certo sentido, as “melhores soluções possíveis” dentro de cada formato, mas revelam características diferentes de erro: a primeira reflete o comportamento nativo da aritmética Posit, enquanto a segunda explicita os efeitos da conversão a partir de uma representação mais refinada.

Essas duas formas de aproximação são ilustradas graficamente nas Figuras 20 y 21. A Cuadro 3 apresenta os erros quadráticos médios observados para cada uma das alternativas, permitindo avaliar o limite inferior de erro introduzido apenas pela escolha do sistema numérico.

TABELA 3 – Erro quadrático médio (MSE) entre a solução analítica em Float64 e as versões exatas reconstruídas diretamente ou convertidas para baixa precisão.

	Posit{16,2}		Posit{8,2}	
	Direto	Convertido	Direto	Convertido
eq1	$1.57 \times 10^{-8}$	$1.62 \times 10^{-9}$	$3.11 \times 10^{-4}$	$1.03 \times 10^{-4}$
eq2	$9.30 \times 10^{-8}$	$1.08 \times 10^{-9}$	$1.38 \times 10^{-3}$	$7.31 \times 10^{-5}$

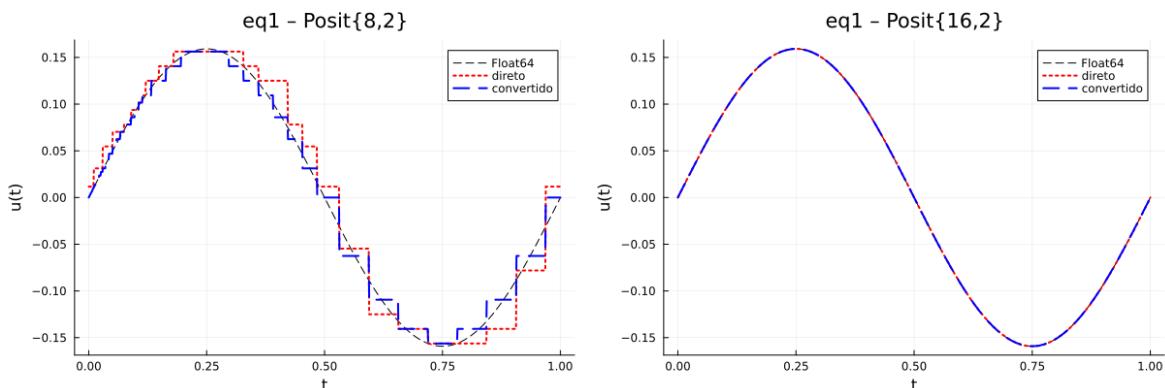


FIGURA 20 – Melhores aproximações para a solução da eq1 com diferentes configurações de Posit.

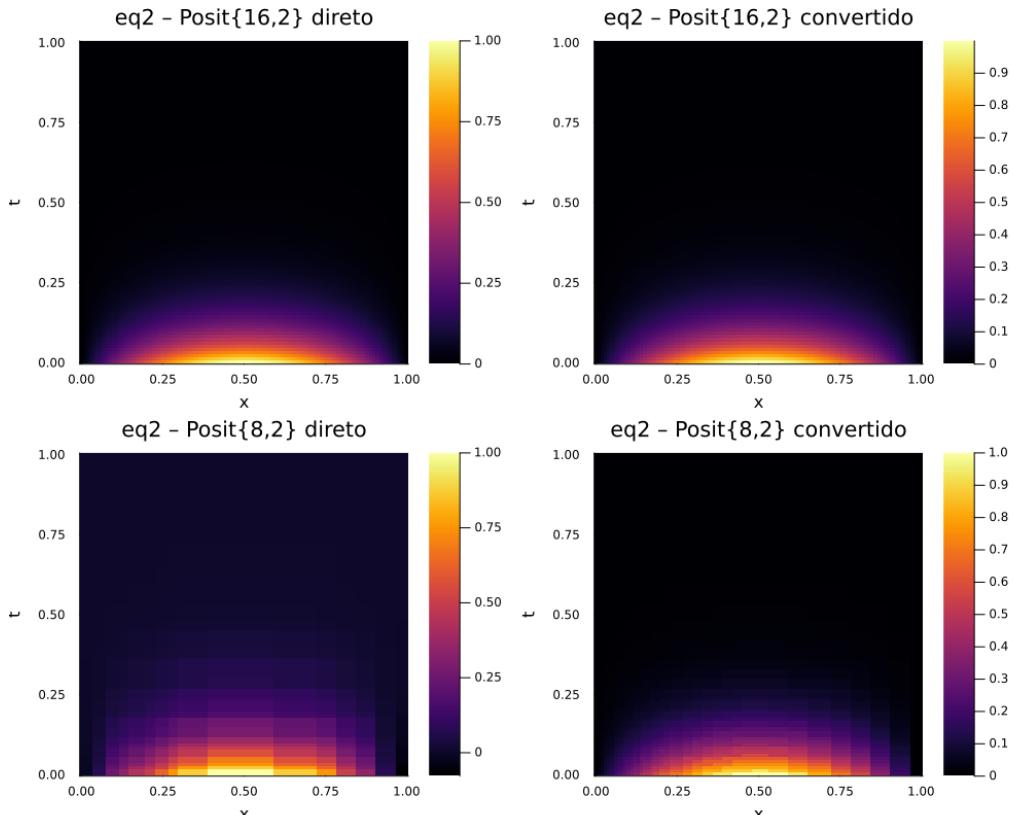


FIGURA 21 – Melhores aproximações para a solução da eq2 com diferentes configurações de Posit.

Para cada equação diferencial avaliada, foram realizados testes numéricos com diferentes arquiteturas de redes neurais e comparações entre os resultados obtidos com e sem o uso da biblioteca NeuralPDE.jl. As redes avaliadas são descritas na Cuadro 4. Para uma referência rápida da localização dos resultados obtidos, consulte a Cuadro 5.

TABELA 4 – Configurações das redes neurais utilizadas nos experimentos

Configuração	Camadas	Neurônios
config1	1	5
config2	1	8
config3	2	5

TABELA 5 – Localização dos resultados dos experimentos

<b>Configuração</b>	<b>Equação</b>	<b>Tabela</b>	<b>Figuras</b>
config1	eq1	Cuadro 6	Figuras 23 y 24
	eq2	Cuadro 9	Figura 30
config2	eq1	Cuadro 7	Figuras 25 y 26
	eq2	Cuadro 10	Figura 31
config3	eq1	Cuadro 8	Figuras 27 y 28
	eq2	Cuadro 11	Figura 32

Os experimentos da eq1 foram repetidos para diferentes números de épocas (*epochs*, nas legendas das figuras), e os resultados foram registrados utilizando as precisões Posit{16,2} e Posit{8,2}. Em todos os testes, a rede foi inicialmente treinada com precisão Posit{16,2} para depois ser convertida em Posit{8,2}, de forma a evitar problemas de saturação de gradiente nas fases iniciais do aprendizado. Essa decisão foi fundamentada na discussão apresentada na Apartado 3.1, que ressalta os riscos do uso direto de baixa precisão sem um período de estabilização. Após o *warmup*, os parâmetros obtidos foram convertidos para o formato Posit{8,2} e o treinamento foi retomado com a nova precisão, buscando aliar a estabilidade numérica inicial à eficiência computacional final. A semente do gerador de números pseudo-aleatórios foi fixada em `rng = MersenneTwister(1234)` a fim de possibilitar reproduzibilidade.

Além disso, realizou-se um experimento adicional em que todo o treinamento foi conduzido exclusivamente com a precisão Posit{8,2}, sem etapas intermediárias em maior precisão. Em todos os casos, foram utilizadas 200 épocas de treinamento. Os resultados, apresentados na Figura 22, evidenciam que a rede treinada diretamente com baixa precisão apresenta dificuldades em capturar adequadamente o comportamento da solução exata. Observa-se que, já nas primeiras épocas, a rede converge rapidamente para um mínimo, estabilizando sua saída por volta da época 25, para config1 e config2. No caso da config3, nota-se uma boa aproximação para  $t \approx 0$  já nas primeiras épocas (cerca de 9 épocas); contudo, essa convergência inicial não se traduz em uma solução globalmente precisa, resultando em uma aproximação deficiente. De forma geral, esses achados reforçam que a adoção de um *warmup* em maior precisão pode ser uma estratégia eficaz para mitigar tais limitações.

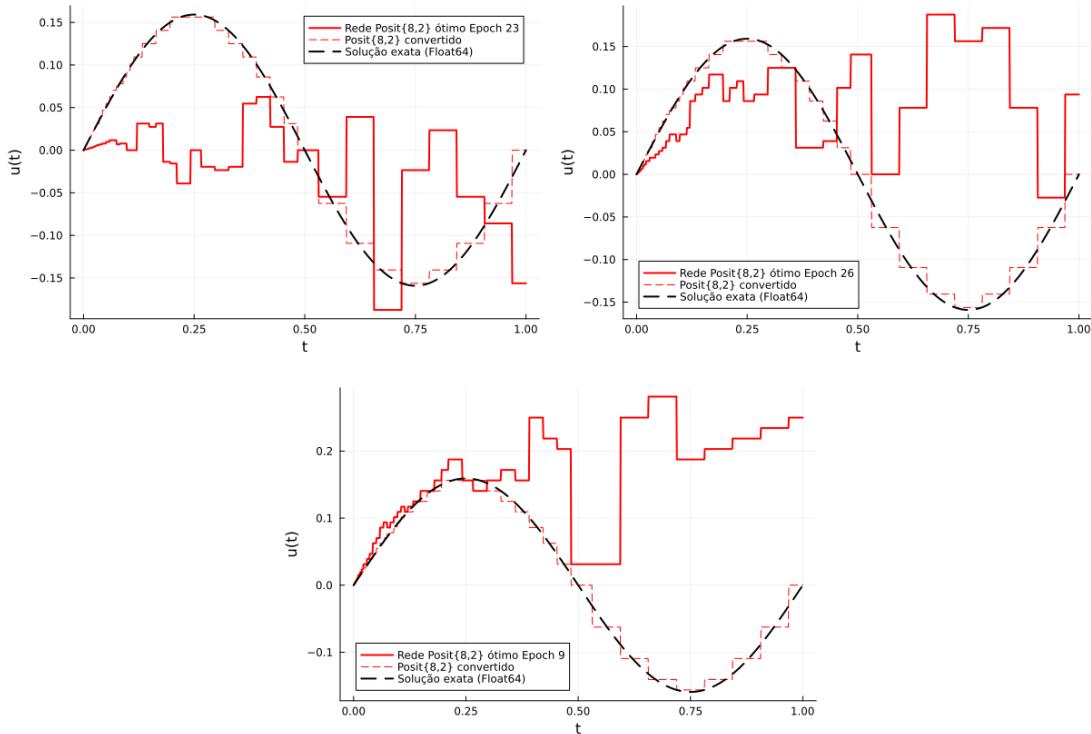


FIGURA 22 – Treinamento direto com Posit{8,2} com config1, config2 e config3, respectivamente, com 200 épocas cada.

Na versão sem o uso da biblioteca NeuralPDE.jl, os otimizadores foram escolhidos levando em consideração as limitações numéricas impostas por cada formato. Para Posit{16,2}, utilizou-se o otimizador Adam com parâmetros  $\eta = 0.01$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  e  $\varepsilon = 10^{-8}$ . Já para Posit{8,2}, optou-se pelo método do gradiente descendente estocástico (SGD), com taxa de aprendizado  $\eta = 0.09$ . A escolha do SGD, nesse caso, deve-se ao fato de que os valores tradicionais dos hiperparâmetros  $\beta_1$  e  $\beta_2$  não possuem representações precisas em Posit{8,2}, o que leva a instabilidades numéricas devidas a divisões por zero durante o cálculo das médias móveis no Adam. Além disso, embora taxas de aprendizado pequenas sejam representáveis nesse formato, elas frequentemente resultam em saturação do gradiente, impedindo atualizações efetivas nos parâmetros da rede.

Vale destacar que, para fins de comparação, os treinamentos foram realizados com um número relativamente pequeno de épocas. Essa escolha se justifica pela observação de que, em aritmética de baixa precisão, as redes tendem a convergir rapidamente para soluções aproximadas quando o problema é formulado sem o uso da biblioteca NeuralPDE.jl. Nessa abordagem direta, as condições de contorno podem ser incorporadas explicitamente por meio de construções como `trial_solution`, o que guia a rede desde o início para regiões do espaço de funções que já satisfazem as restrições do problema. Em contraste, ao utilizar a NeuralPDE.jl, as condições de contorno são tratadas como termos adicionais na função de perda, e sua convergência depende do processo de otimização. Como consequência, geralmente são necessárias

mais épocas para que a rede aprenda simultaneamente uma solução que atenda a equação diferencial e cumpra as condições de contorno.

Adicionalmente, nos experimentos com a NeuralPDE.jl, não foi aplicada a estratégia de refinamento com  $\text{Posit}\{8,2\}$  após o treino inicial com  $\text{Posit}\{16,2\}$ , tendo em vista a maior complexidade e a instabilidade observadas ao realizar essa transição dentro do framework automático. Dessa forma, todos os resultados com NeuralPDE.jl foram obtidos diretamente com  $\text{Posit}\{16,2\}$ .

TABELA 6 – eq1-config1

<b>Épocas</b>		<b>Com NeuralPDE</b>	<b>Sem NeuralPDE</b>	
$\text{Posit}\{16,2\}$	$\text{Posit}\{8,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{8,2\}$
50	100	6,236328125	4,00390625	1,5
100	50	4,845703125	2,34375	1,125
200	50	3,796875	0,843017578	2,25
500	50	1,9775390625	0,036712646	0,8125
2 000	100	0,2901611328125	0,007118225	1,0

TABELA 7 – eq1-config2

<b>Épocas</b>		<b>Com NeuralPDE</b>	<b>Sem NeuralPDE</b>	
$\text{Posit}\{16,2\}$	$\text{Posit}\{8,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{8,2\}$
50	100	1,7412109375	1,081542969	0,5625
100	50	0,8740234375	0,098815918	0,40625
200	50	0,375732421875	0,022262573	1,5
500	50	0,01837158203125	0,019607544	1,125
2000	100	0,00464630126953125	0,001556396	2,25

TABELA 8 – eq1-config3

<b>Épocas</b>		<b>Com NeuralPDE</b>	<b>Sem NeuralPDE</b>	
$\text{Posit}\{16,2\}$	$\text{Posit}\{8,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{16,2\}$	$\text{Loss}\{8,2\}$
50	100	1,27392578125	0,547607422	2,25
100	50	0,804443359375	0,073425293	0,5625
200	50	0,288330078125	0,025421143	1,125
500	50	0,0163116455078125	0,017883301	1,5
2000	100	0,00443267822265625	0,000736237	0,6875

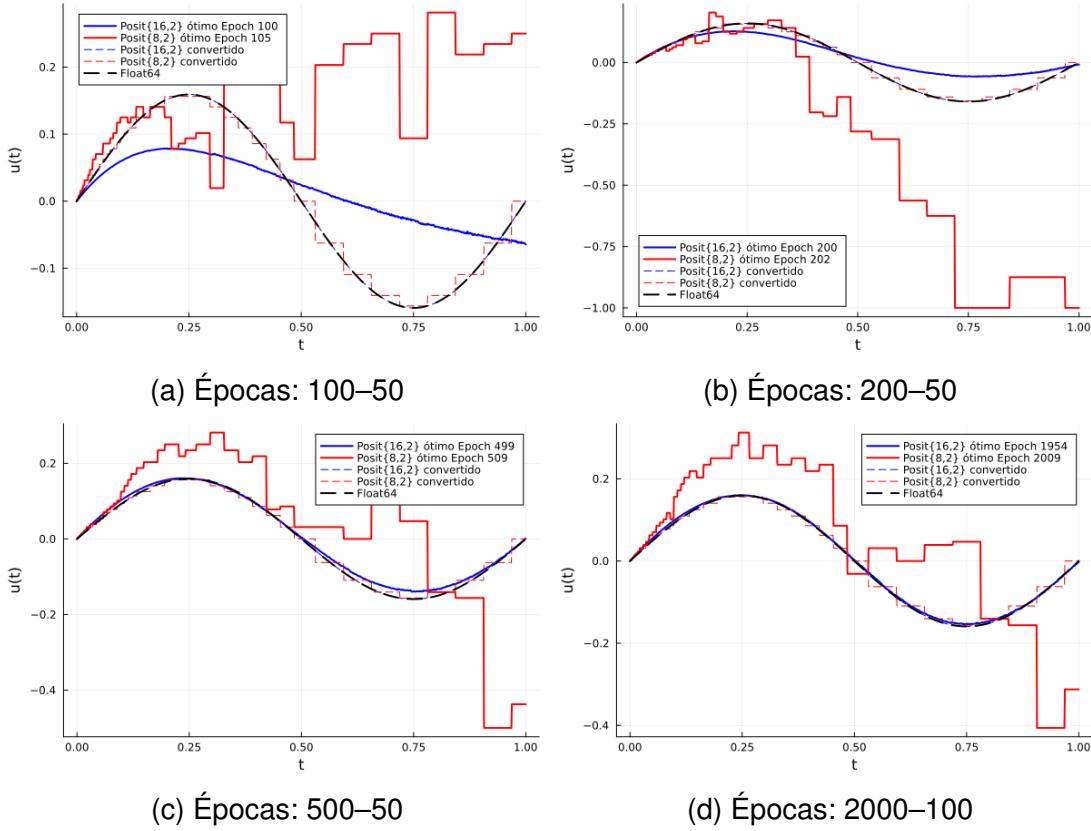


FIGURA 23 – Melhores soluções da rede eq1-config1, sem o uso da biblioteca NeuralPDE.jl, obtido com Posit{16,2} seguido de refinamento com Posit{8,2}.

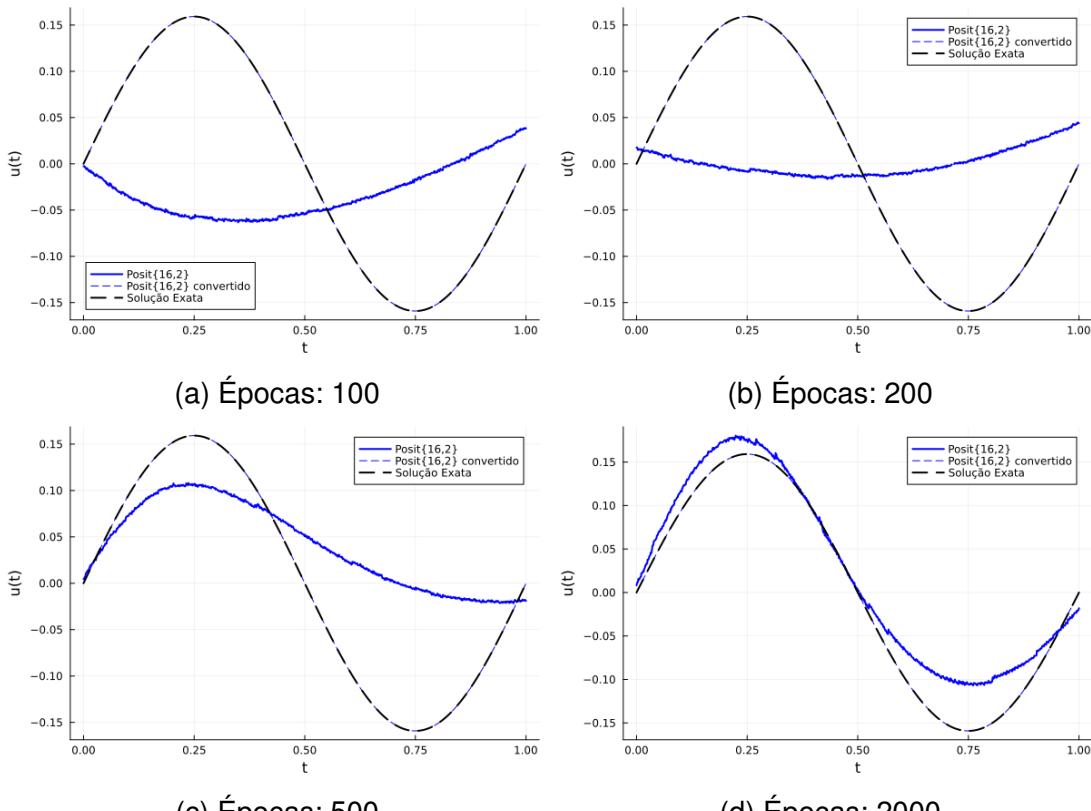


FIGURA 24 – Melhores soluções da rede eq1-config1, com o uso da biblioteca NeuralPDE.jl, obtido com Posit{16,2} seguido de refinamento com Posit{8,2}.

Observa-se que para resolver a eq1 na config1, mesmo um número reduzido de épocas já é suficiente para produzir aproximações satisfatórias com Posit{16,2}, considerando as limitações impostas pela precisão numérica utilizada. No entanto, as soluções obtidas em Posit{8,2}, mesmo após o *warmup*, ainda apresentam limitações na qualidade geral do ajuste. Destaca-se, contudo, a boa aproximação obtida para a função solução na vizinhança de  $t \approx 0$ , mesmo utilizando Posit{8,2}.

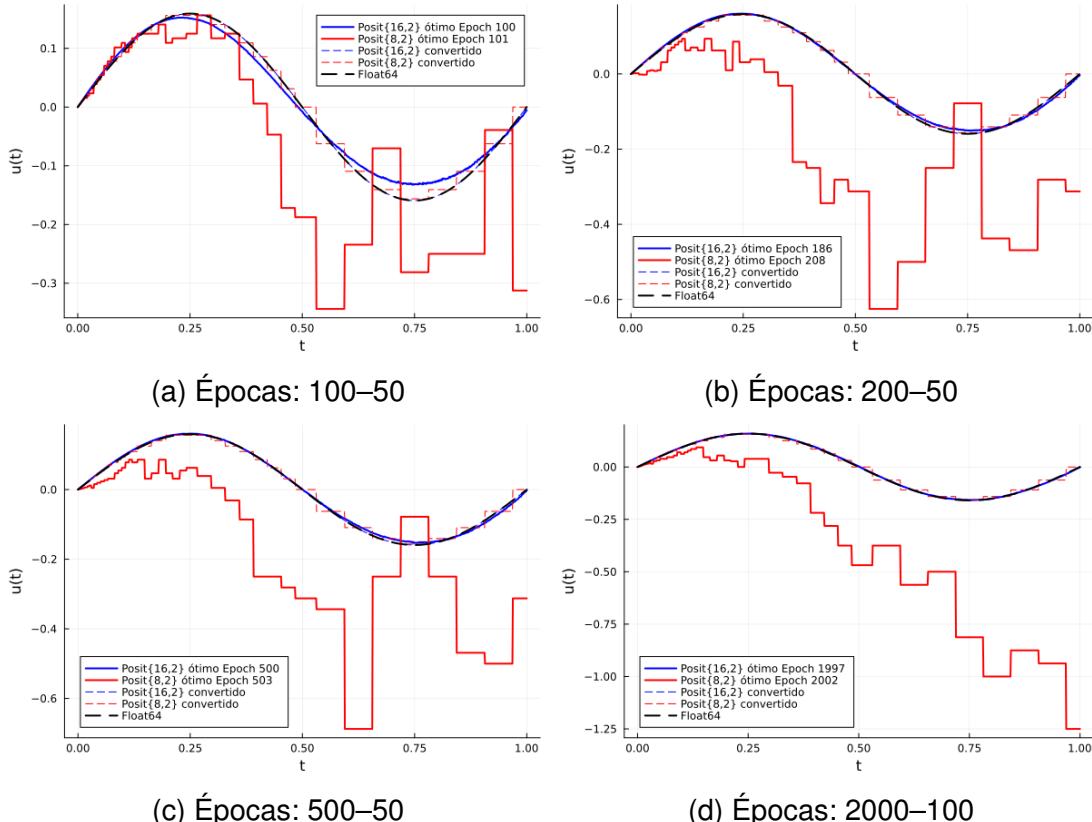


FIGURA 25 – Melhores soluções da rede eq1-config2, sem o uso da biblioteca NeuralPDE.jl, obtido com Posit{16,2} seguido de refinamento com Posit{8,2}.

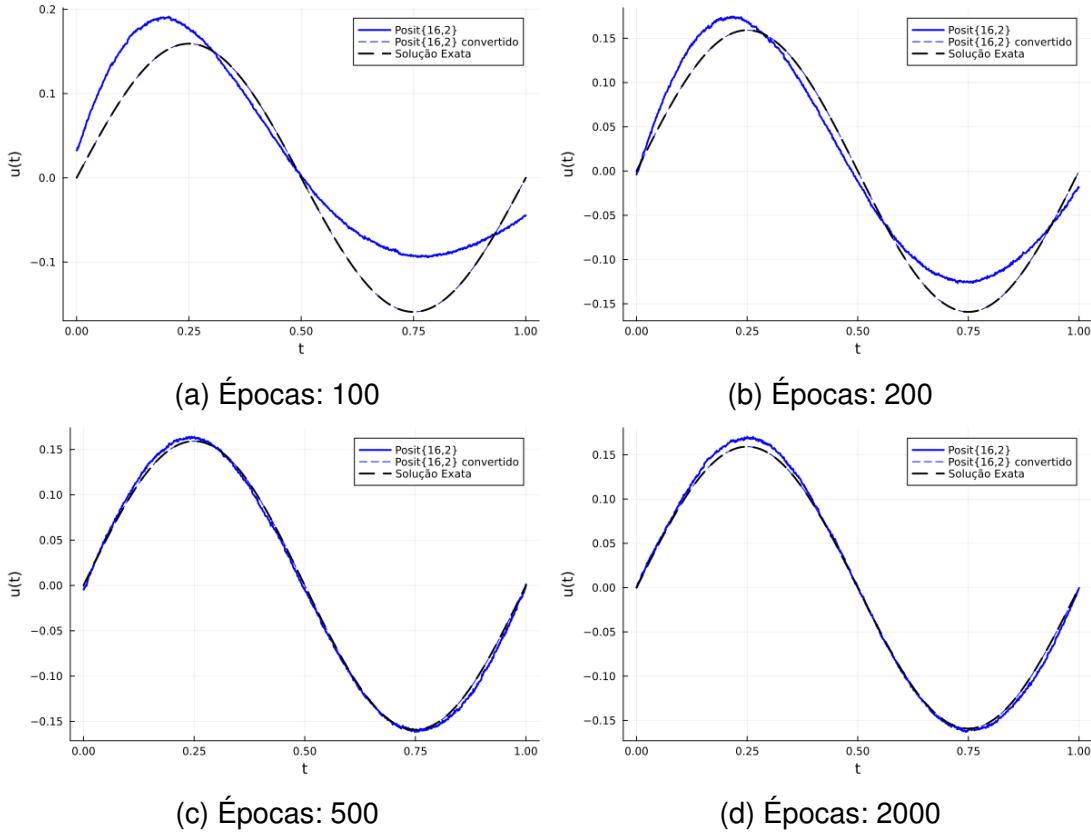


FIGURA 26 – Melhores soluções da rede eq1-config2, com o uso da biblioteca NeuralPDE.jl, obtido com  $\text{Posit}\{16,2\}$  seguido de refinamento com  $\text{Posit}\{8,2\}$ .

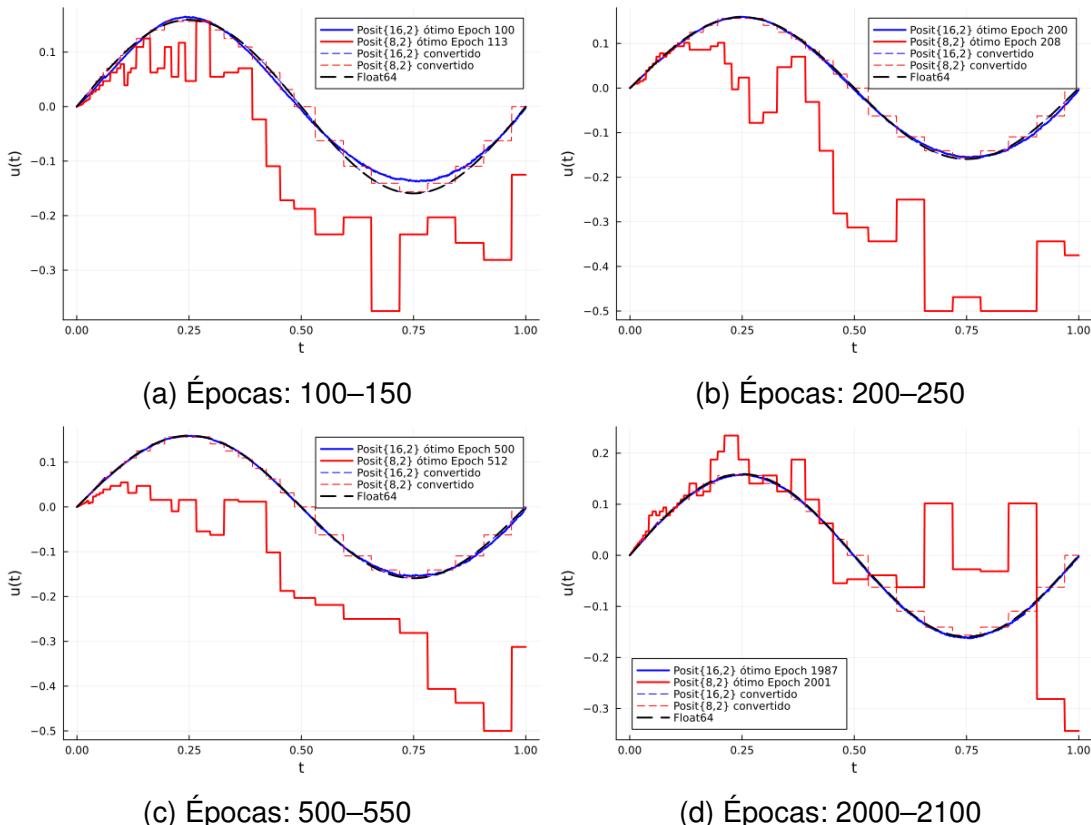


FIGURA 27 – Melhores soluções da rede eq1-config3, sem o uso da biblioteca NeuralPDE.jl, obtido com  $\text{Posit}\{16,2\}$  seguido de refinamento com  $\text{Posit}\{8,2\}$ .

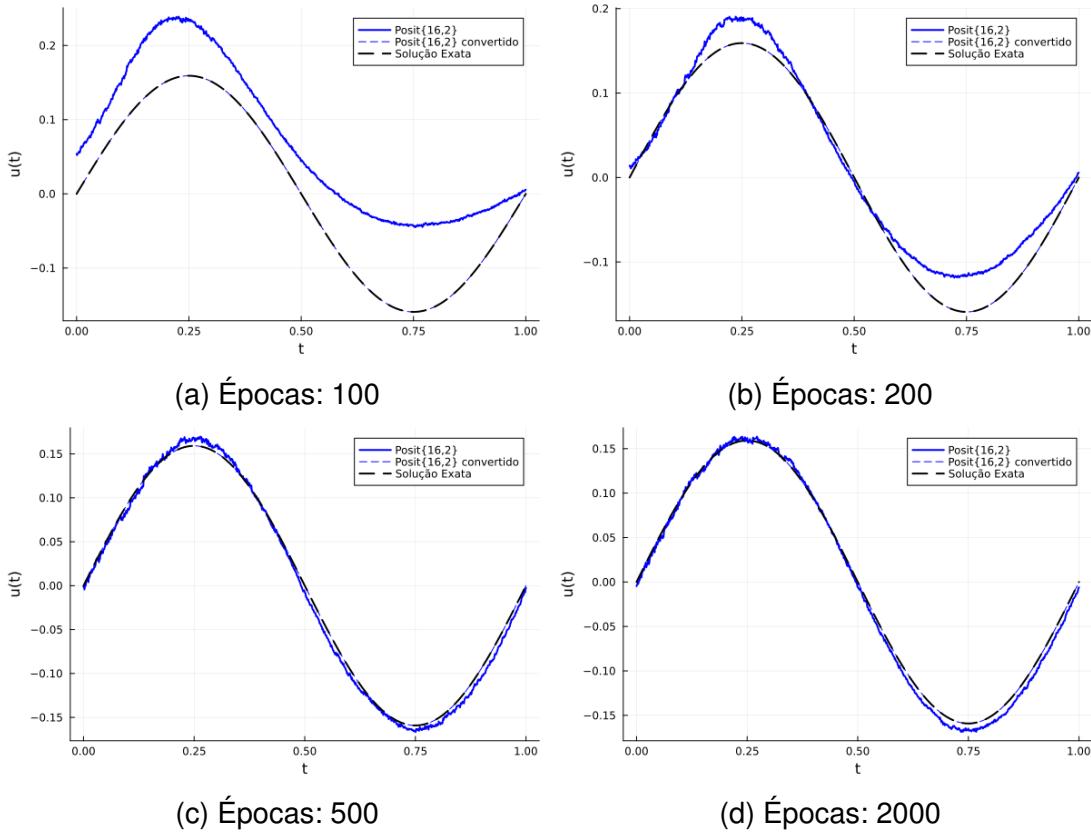


FIGURA 28 – Melhores soluções da rede eq1-config3, com o uso da biblioteca NeuralPDE.jl, obtido com Posit{16,2} seguido de refinamento com Posit{8,2}.

Para a eq2, optou-se por realizar todo o treinamento diretamente no formato  $\text{Posit}\{16, 2\}$ , uma vez que o domínio do problema não apresenta valores concentrados próximos de zero. Embora fosse possível adotar um *warmup* em  $\text{Posit}\{32, 2\}$ , essa alternativa foi descartada devido à ausência de hardware com suporte nativo à aritmética Posit e de uma biblioteca paralelizável, o que tornaria o tempo de treinamento inviável. Assim como na eq1, as condições de contorno e inicial são impostas de forma forte, e o treinamento é conduzido sem o uso da biblioteca NeuralPDE.jl, apenas para os pontos interiores do domínio, como mostra a Figura 29. Nesta configuração, são avaliados cenários com 200, 500 e 1000 épocas.

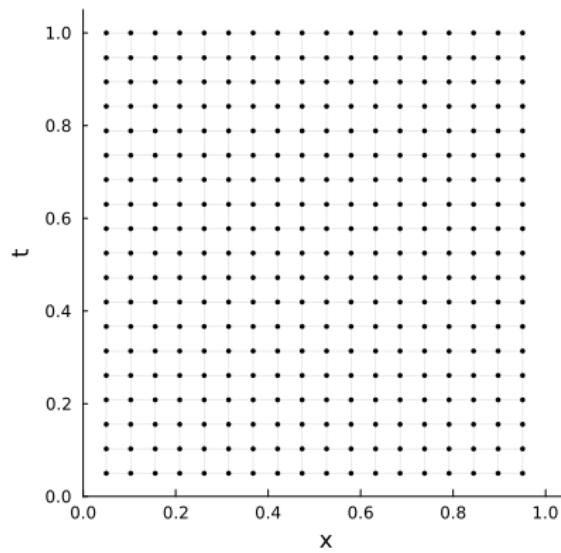


FIGURA 29 – Distribuição dos pontos de avaliação com Posit{16,2}, gerados a partir de uma malha uniforme no intervalo  $(0, 1)$  para  $x$  e  $t$ , excluindo as fronteiras, utilizados para avaliar a rede durante o treinamento.

TABELA 9 – eq2-config1 sem NeuralPDE.jl.

<b>Épocas</b>	<b>Loss{16,2}</b>
200	30,671875
500	13,234375
1 000	6,173828125

TABELA 10 – eq2-config2 sem NeuralPDE.jl.

<b>Épocas</b>	<b>Loss{16,2}</b>
200	18,203125
500	8,953125
1 000	4,83203125

TABELA 11 – eq2-config3 sem NeuralPDE.jl.

<b>Épocas</b>	<b>Loss{16,2}</b>
200	17,90625
500	7,189453125
1 000	4,173828125

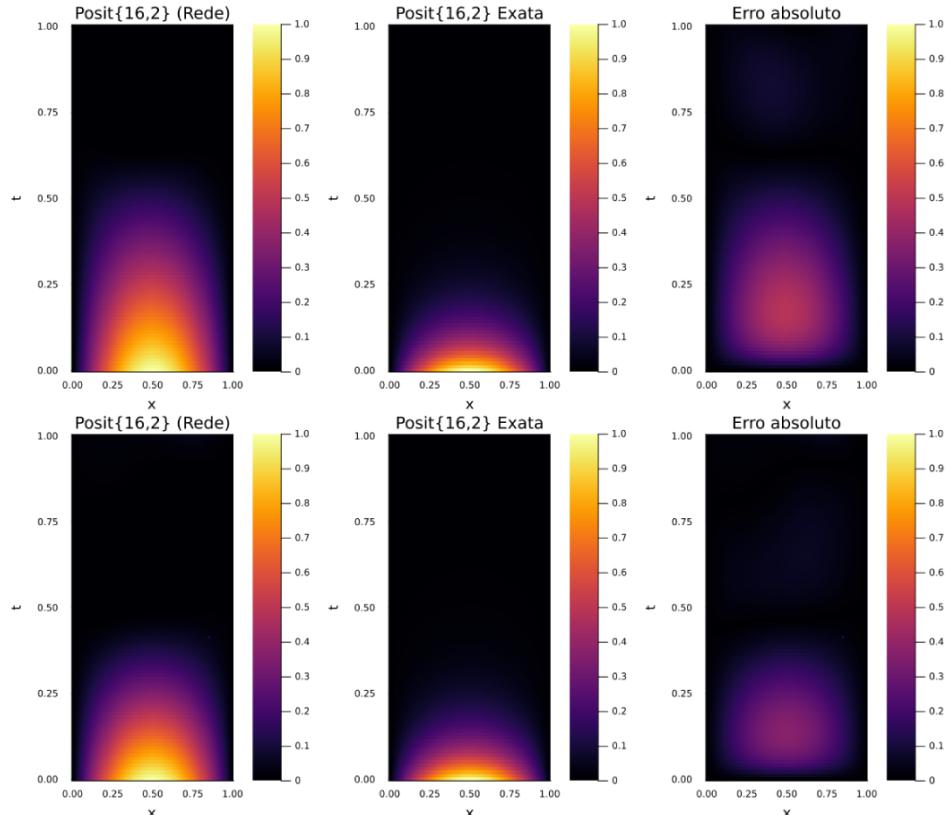


FIGURA 30 – Resultados obtidos para a eq2-config1 com Posit{16,2} utilizando 500 e 1000 épocas, respectivamente.

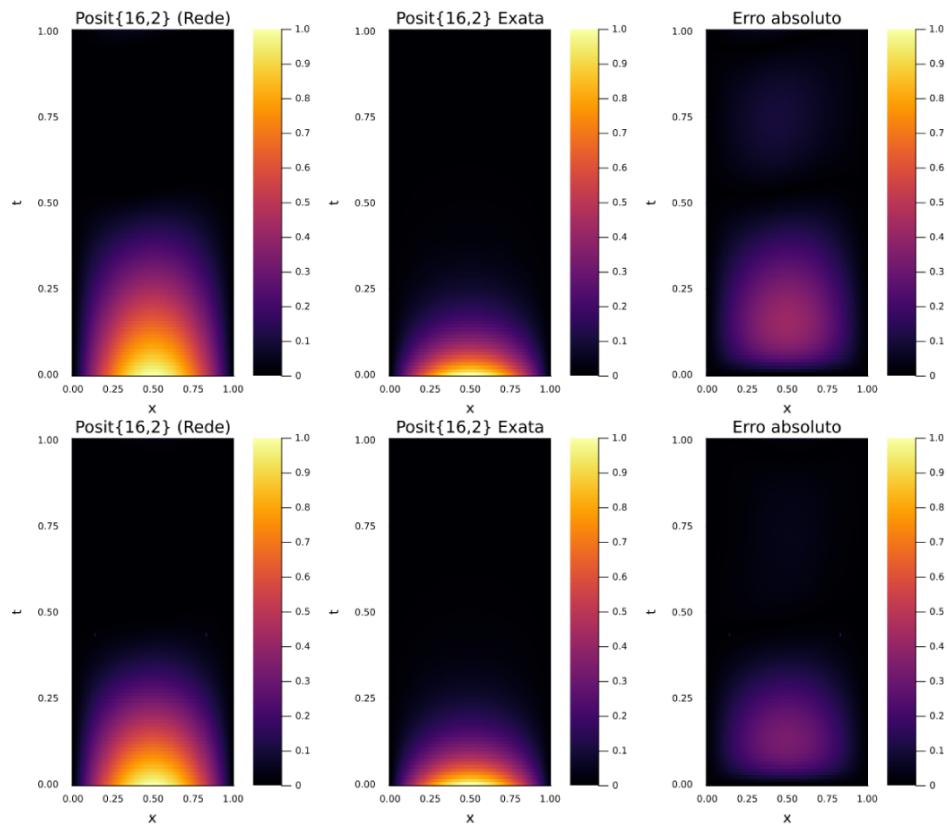


FIGURA 31 – Resultados obtidos para a eq2-config2 com Posit{16,2} utilizando 500 e 1000 épocas, respectivamente.

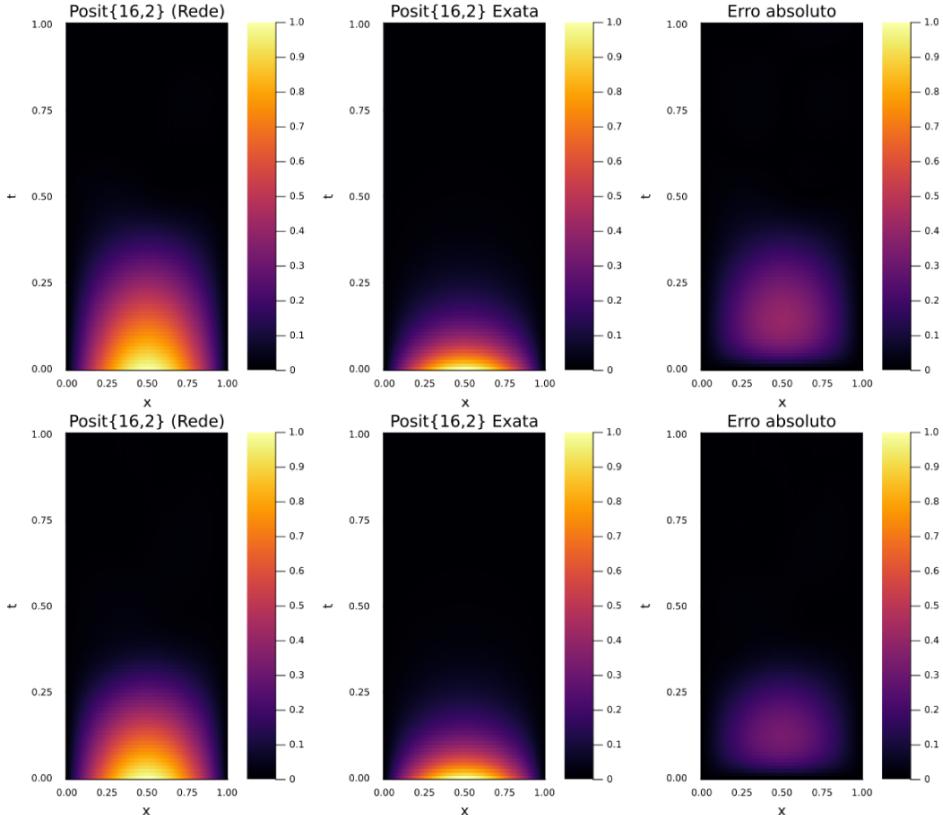


FIGURA 32 – Resultados obtidos para a eq2-config3 com Posit{16,2} utilizando 500 e 1000 épocas, respectivamente.

Para as três arquiteturas de rede consideradas, observa-se que a solução está de fato convergindo, no sentido de que a *loss* diminui a cada iteração, embora de forma relativamente lenta. Como pode ser visto nas Figuras 30 a 32, ainda há regiões no interior do domínio em que a aproximação não é satisfatória. Apesar disso, a *loss* avaliada ponto a ponto não apresenta valores elevados; entretanto, quando analisamos o erro quadrático acumulado, sua soma atinge valores na ordem de dezenas, conforme indicado nas Quadros 9 a 11.

É importante destacar que existem estratégias conhecidas para reduzir a *loss* e melhorar a qualidade da solução em regiões críticas, como o uso de malhas não uniformes no domínio – concentrando mais pontos onde se espera maior variação da solução – ou a atribuição de pesos diferenciados na função de perda para pontos próximos de  $t = 0$ . Tais abordagens podem aumentar a precisão local e acelerar a convergência em problemas específicos. No entanto, essas técnicas não constituem o foco do presente estudo, cujo objetivo principal é avaliar o comportamento do sistema numérico Posit no treinamento de PINNs, e não otimizar individualmente o desempenho da rede para classes específicas de EDPs.

## 6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, investigamos o uso do sistema numérico Posit como alternativa ao padrão IEEE-754 em tarefas de resolução de equações diferenciais ordinárias (EDOs) e parciais (EDPs) por meio de *Physics-Informed Neural Networks* (PINNs). Para viabilizar essa investigação, foi desenvolvida do zero a biblioteca PositPR.jl, uma contribuição inédita em código aberto que oferece suporte completo ao sistema Posit, incluindo operações aritméticas fundamentais, funções matemáticas e integração com bibliotecas modernas do ecossistema Julia voltadas ao aprendizado de máquina e solução de EDPs, como Lux.jl e NeuralPDE.jl.

Ao preencher uma lacuna concreta na infraestrutura disponível para pesquisa com formatos numéricos alternativos, este trabalho não apenas tornou possíveis os experimentos realizados, mas também posiciona a biblioteca PositPR.jl como uma ferramenta versátil e reproduzível para estudos futuros. Trata-se, portanto, de uma contribuição que vai além do estudo empírico e estabelece um novo ponto de partida para o uso prático do sistema Posit em ambientes de simulação numérica, com potencial impacto em áreas como inteligência artificial de baixa potência, computação científica e otimização energética.

Os testes realizados evidenciaram que, mesmo com redes neurais pequenas (poucos neurônios e camadas) e número limitado de épocas de treinamento, foi possível alcançar soluções bem aproximadas para EDOs e EDPs clássicas. Em muitos casos, os resultados obtidos com Posit apresentaram boa concordância com soluções de referência, mesmo utilizando apenas 16 bits de precisão. Isso sugere que o uso do Posit pode representar uma via promissora para reduzir o custo computacional sem comprometer a qualidade da solução – sobretudo em cenários onde recursos computacionais são escassos ou a escalabilidade do modelo é um fator limitante.

A estratégia de iniciar o treinamento em  $\text{Posit}\{16, 2\}$  e posteriormente realizar uma conversão para formatos de menor precisão não mostrou-se eficaz para estabilizar a otimização e acelerar a convergência. O uso direto de  $\text{Posit}\{8, 2\}$  também demonstrou instabilidades fora da vizinhança da origem, o que indica que seu uso deve ser feito com cautela. Ainda assim, nos casos em que a função ou o domínio está concentrado próximo de zero,  $\text{Posit}\{8, 2\}$  pode oferecer desempenho adequado com menor custo. O *warmup* pode ser útil quando feita em precisão maior, como  $\text{Posit}\{32, 2\}$ .

Este trabalho contribui também para uma discussão mais ampla sobre os limites da representação numérica em ambientes de aprendizado de máquina. Em um cenário onde a eficiência computacional se tornou tão crítica quanto a acurácia,

a possibilidade de treinar redes com formatos numéricos adaptativos – como o Posit – pode representar um avanço considerável em termos de desempenho, economia energética e portabilidade.

Algumas direções naturais para trabalhos futuros incluem:

- Publicação e manutenção da biblioteca PositPR.jl: a disponibilização da biblioteca em um repositório público oficial, acompanhada de documentação completa, exemplos de uso e testes unitários, permitirá sua adoção pela comunidade. Além disso, ajustes de compatibilidade com versões recentes do ecossistema Julia e integração em registries facilitarão sua utilização em pesquisas e aplicações práticas.
- Otimização da biblioteca PositPR.jl para redes neurais: embora a biblioteca desenvolvida seja funcional e flexível, ainda há espaço para otimizações significativas, especialmente no contexto de redes neurais. Melhorias em desempenho computacional, redução de alocações e suporte a operações vetorizadas podem tornar o uso do sistema Posit mais competitivo em aplicações práticas de aprendizado de máquina.
- Implementação em hardware ou simulação de baixo nível: como os ganhos do sistema Posit são mais expressivos em arquiteturas computacionais que exploram sua estrutura nativa, uma possível linha de desenvolvimento envolve a simulação de desempenho energético e a adaptação da biblioteca para execução em ambientes com suporte a Posit em hardware, como *Field Programmable Gate Arrays (FPGAs)*, *Application Specific Integrated Circuits (ASICs)* ou *Tensor Processing Units (TPUs)*.
- Estudo de modelos mais complexos: aplicar o sistema Posit em arquiteturas mais elaboradas – como redes convolucionais, modelos com entradas multivariadas, domínios irregulares ou equações diferenciais com maior rigidez – pode revelar novas limitações ou vantagens em relação aos formatos tradicionais.
- Exploração de aplicações com comportamento local sensível: o formato Posit apresenta maior resolução numérica em torno da origem. Assim, sua aplicação em problemas cujo domínio ou solução esteja concentrado próximo de zero (como certos modelos físicos, biomédicos ou financeiros) pode oferecer vantagens relevantes, tanto em termos de precisão quanto de eficiência numérica.

Concluímos, portanto, que a adoção do sistema numérico Posit no treinamento de redes neurais para solução numérica de EDOs e EDPs é promissora. O trabalho realizado oferece uma base sólida para avanços aplicados na interseção entre representação numérica e aprendizado de máquina.

## REFERÊNCIAS

- BOX, G. E. P.; MULLER, M. E. A Note on the Generation of Random Normal Deviates. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 29, n. 2, p. 610–611, 1958. DOI: [10.1214/aoms/1177706645](https://doi.org/10.1214/aoms/1177706645). Citado 1 vez na página 21.
- CARMICHAEL, Z.; LANGROUDI, H. F.; KHAZANOV, C.; LILLIE, J.; GUSTAFSON, J. L.; KUDITHIPUDI, D. Deep Positron: A Deep Neural Network Using the Posit Number System. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). [S.I.: s.n.], 2019. P. 1421–1426. DOI: [10.23919/DATE.2019.8715262](https://doi.org/10.23919/DATE.2019.8715262). Citado 1 vez na página 8.
- DIXIT, V. K.; RACKAUCKAS, C. **Optimization.jl: A Unified Optimization Package**. [S.I.]: Zenodo, mar. 2023. DOI: [10.5281/zenodo.7738525](https://doi.org/10.5281/zenodo.7738525). Disponível em: <https://doi.org/10.5281/zenodo.7738525>. Citado 2 vezes nas páginas 8, 19.
- GUSTAFSON, J. L.; YONEMOTO, I. T. Beating Floating Point at its Own Game: Posit Arithmetic. **Supercomputing Frontiers and Innovations**, v. 4, n. 2, p. 71–86, 2017. DOI: [10.14529/jsfi170206](https://doi.org/10.14529/jsfi170206). Citado 3 vezes nas páginas 7, 10, 11.
- IEEE. IEEE Standard for Floating-Point Arithmetic. **IEEE Std 754-2019 (Revision of IEEE 754-2008)**, p. 1–84, 2019. DOI: [10.1109/IEEESTD.2019.8766229](https://doi.org/10.1109/IEEESTD.2019.8766229). Citado 1 vez na página 10.
- INNES, M.; SABA, E.; FISCHER, K.; GANDHI, D.; RUDILOSSO, M. C.; JOY, N. M.; KARMALI, T.; PAL, A.; SHAH, V. Fashionable Modelling with Flux. **CoRR**, abs/1811.01457, 2018. arXiv: [1811.01457](https://arxiv.org/abs/1811.01457). Disponível em: <https://arxiv.org/abs/1811.01457>. Citado 1 vez na página 8.
- INNES, M. Flux: Elegant Machine Learning with Julia. **Journal of Open Source Software**, 2018. DOI: [10.21105/joss.00602](https://doi.org/10.21105/joss.00602). Citado 1 vez na página 8.
- JOST, T. T. **Compilation and optimizations for variable precision floating-Point arithmetic: from language and libraries to code generation**. 2021. Université Grenoble Alpes. Computer Arithmetic. NNT: 2021GRALM020. Tel: <https://theses.hal.science/tel-03414534>. Citado 1 vez na página 15.

LU, J.; FANG, C.; XU, M.; LIN, J.; WANG, Z. Evaluations on Deep Neural Networks Training Using Posit Number System. **IEEE Transactions on Computers**, v. 70, n. 2, p. 174–187, 2021. DOI: [10.1109/TC.2020.2985971](https://doi.org/10.1109/TC.2020.2985971). Citado 1 vez na página 8.

LU, J.; LU, S.; WANG, Z.; FANG, C.; LIN, J.; WANG, Z.; DU, L. **Training Deep Neural Networks Using Posit Number System**. [S.I.: s.n.], 2019. arXiv: [1909.03831 \[cs.LG\]](https://arxiv.org/abs/1909.03831). Disponível em: <https://arxiv.org/abs/1909.03831>. Citado 4 vezes nas páginas 7, 16.

PAL, A. **Lux: Explicit Parameterization of Deep Neural Networks in Julia**. [S.I.]: Zenodo, abr. 2023. If you use this software, please cite it as below. DOI: [10.5281/zenodo.7808904](https://doi.org/10.5281/zenodo.7808904). Disponível em: <https://doi.org/10.5281/zenodo.7808904>. Citado 2 vezes nas páginas 8, 18.

PAL, A. **On Efficient Training & Inference of Neural Differential Equations**. 2023. Massachusetts Institute of Technology. Citado 1 vez na página 8.

RACKAUCKAS, C.; MA, Y.; MARTENSEN, J.; WARNER, C.; ZUBOV, K.; SUPEKAR, R.; SKINNER, D.; RAMADHAN, A. Universal differential equations for scientific machine learning. **arXiv preprint arXiv:2001.04385**, 2020. Citado 1 vez na página 19.

RAPOSO, G.; TOMÁS, P.; ROMA, N. Positnn: Training Deep Neural Networks with Mixed Low-Precision Posit. In: ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). [S.I.: s.n.], 2021. P. 7908–7912. DOI: [10.1109/ICASSP39728.2021.9413919](https://doi.org/10.1109/ICASSP39728.2021.9413919). Citado 2 vezes nas páginas 8, 16.

ZUBOV, K.; MCCARTHY, Z.; MA, Y.; CALISTO, F.; PAGLIARINO, V.; AZEGLIO, S.; BOTTERO, L.; LUJÁN, E.; SULZER, V.; BHARAMBE, A.; VINCHHI, N.; BALAKRISHNAN, K.; UPADHYAY, D.; RACKAUCKAS, C. **NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations**. [S.I.]: arXiv, 2021. DOI: [10.48550/ARXIV.2107.09443](https://doi.org/10.48550/ARXIV.2107.09443). Disponível em: <https://arxiv.org/abs/2107.09443>. Citado 2 vezes nas páginas 8, 18.