

Avaliação de Desempenho e Precisão na Implementação de Funções Trigonométricas Aproximadas

João B. Oliveira¹, Rogério A. Gonçalves¹, João Fabrício Filho¹

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87.301-899 – Campo Mourão – PR – Brazil

joliveira.2022@alunos.utfpr.edu.br, {rogerioag, joaof}@utfpr.edu.br

Abstract. *This work evaluates approximate implementations of the floating-point trigonometric functions $\sin$, $\cos$, and $\tan$, comparing performance and precision. Polynomial approximations were implemented and evaluated using the Horner and Estrin methods, along with geometric approximations based on parabolas. The results show that the polynomial models achieve speedups greater than $2\times$ compared to `glibc`, while maintaining approximately ten decimal places of precision. In contrast, the geometric models achieve speedups of up to $10\times$, but with precision reduced to about two decimal places.*

Resumo. *Este trabalho avalia implementações aproximadas das funções trigonométricas $\sin$, $\cos$ e $\tan$ em ponto flutuante, comparando desempenho e precisão. Foram implementadas aproximações polinomiais, avaliadas pelos métodos de Horner e Estrin, além de aproximações geométricas baseadas em parábolas. Os resultados mostram que os modelos polinomiais alcançam ganhos superiores a $2\times$ em relação à `glibc`, mantendo aproximadamente dez casas decimais de precisão. Já os modelos geométricos atingem ganhos de até $10\times$, porém com precisão reduzida para cerca de duas casas decimais.*

1. Introdução

Operações em ponto flutuante são essenciais em áreas como computação gráfica, simulações físicas, processamento de sinais e aprendizado de máquina. Embora o padrão IEEE-754 permita representar uma ampla faixa de valores, nem todos os números reais podem ser armazenados exatamente com uma quantidade finita de bits [Goldberg 1991]. Por isso, operações aritméticas e funções matemáticas estão sujeitas a arredondamentos e pequenos erros numéricos.

Bibliotecas matemáticas tradicionais, como a `glibc`¹, priorizam alta precisão e robustez para diferentes faixas de entrada. Esta característica pode aumentar o custo computacional de funções como $\sin$, $\cos$ e $\tan$. Em aplicações tolerantes a pequenos erros, como renderização gráfica, jogos, simulações aproximadas e processamento multimídia, nem sempre é necessário utilizar a implementação mais precisa disponível.

Essa tolerância permite explorar funções trigonométricas aproximadas, sacrificando parte da precisão em troca de maior desempenho. A ideia também aparece em

¹<https://www.gnu.org/software/libc/>

bibliotecas otimizadas, como a AOCL-LibM da AMD, que oferece versões com maior desempenho e menor precisão [Advanced Micro Devices, Inc. 2026].

Outros trabalhos exploram estratégias para aprimorar funções matemáticas mantendo controle sobre o erro numérico. A RLIBM [Aanjaneya and Nagarakatte 2023], por exemplo, aproxima diretamente os resultados já arredondados, em vez de aproximar a curva real da função. Com isso, seu objetivo é garantir que o valor calculado pertença ao intervalo que produz o arredondamento correto.

Outra abordagem é o OpTuner [Briggs and Panchekha 2022], uma ferramenta que seleciona automaticamente a implementação mais adequada de cada função matemática. Essa escolha considera o domínio de entrada e a tolerância ao erro, buscando a alternativa que oferece melhor desempenho sem comprometer a qualidade exigida pela aplicação. Contudo, funções trigonométricas para ponto flutuante ainda são funções que permitem exploração por diferentes formas de aproximação.

Este trabalho avalia diferentes aproximações para funções trigonométricas em ponto flutuante, comparando o ganho de desempenho com a perda de precisão introduzida. Foi implementada uma biblioteca chamada MathX, contendo diferentes versões aproximadas das funções `sin`, `cos` e `tan`. As implementações usam aproximações polinomiais e geométricas, com o objetivo de avaliar diferentes pontos de equilíbrio entre desempenho e precisão. Nas versões polinomiais, os coeficientes foram gerados com a ferramenta Sollya [Chevallard et al. 2010]. Além disso, foram implementadas aproximações geométricas baseadas em parábolas, que usam expressões mais simples e rápidas, mas tendem a apresentar maior erro numérico. A avaliação experimental mostra *speedups* de até $10\times$ com erro médio de 534007596000074.2 ULPs.

2. Materiais e Métodos

Esta seção descreve alguns parâmetros de implementação e da avaliação das funções implementadas na biblioteca MathX. Antes da avaliação de cada polinômio, a entrada é reduzida para o intervalo $[-\frac{\pi}{4}, \frac{\pi}{4}]$. Essa redução é importante porque funções trigonométricas são periódicas, o que permite aproximar a função em um intervalo menor e mais controlado.

Foram avaliadas duas formas de cálculo dos polinômios: *Horner* e *Estrin*. O método de *Horner* reduz a quantidade de operações ao reescrever o polinômio em forma fatorada. Já o método de *Estrin* busca expor mais paralelismo entre instruções, o que pode favorecer a execução em processadores modernos.

Os experimentos foram conduzidos em um sistema com processador AMD Ryzen 5, com 6 *cores* e 12 *threads*, 16 GB de memória DDR4, Sistema Operacional GNU/Linux na distribuição Fedora 43 e compilador Clang com as *flags* `-O3 -march=native`.

As funções foram implementadas e avaliadas para o tipo *double*. Foram utilizados 4096 valores de entrada, gerados aleatoriamente a partir de uma semente fixa. As técnicas foram comparadas com as versões da `glibc` 2.42. Os intervalos de teste seguiram os mesmos definidos pela CORE-MATH [Sibidanov et al. 2022]: $[-\frac{\pi}{2}, \frac{\pi}{2}]$ para `sin` e `tan`, e $[0, 2\pi]$ para `cos`. O tempo de execução foi medido com a biblioteca *Google Benchmark* [Google 2013]. A precisão foi avaliada por meio da diferença em *Units in*

the Last Place (ULP), que indica quantos valores representáveis em ponto flutuante existem entre o resultado da implementação de referência e o resultado da implementação aproximada. O código das implementações e os modelos utilizados estão disponíveis no repositório do projeto MathX².

3. Resultados e Análise

A Figura 1 apresenta a média do tempo de CPU gasto por cada técnica avaliada. As funções padrão da `glibc` demandaram mais tempo de execução do que as aproximações implementadas. As versões baseadas em Horner e Estrin apresentaram desempenho semelhante, com ganhos de aproximadamente $2.3\times$ em relação à biblioteca padrão. Para a função `tan`, esse ganho chegou a cerca de $4\times$, esse resultado é explicado devido a necessidade de maiores polinômios para representar os seus resultados. As versões geométricas obtiveram os melhores resultados de desempenho, alcançando ganhos de até $10\times$.

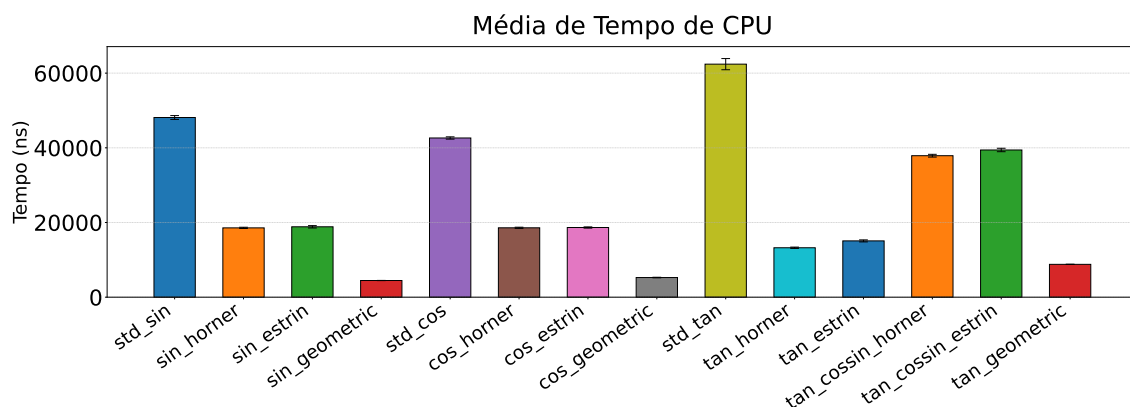


Figura 1. Tempo de CPU das funções avaliadas e suas respectivas técnicas.

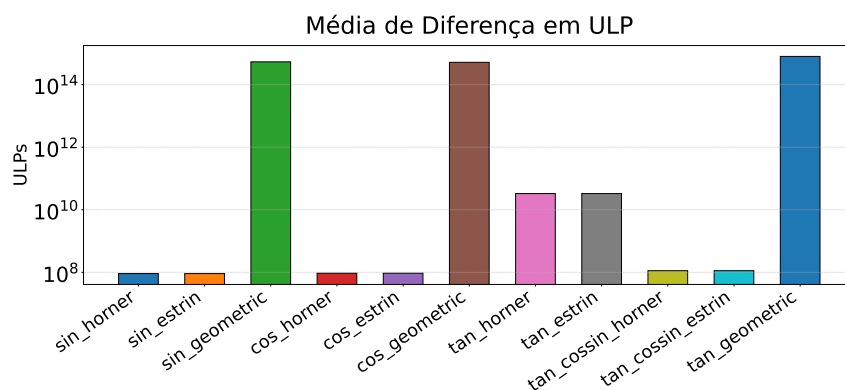


Figura 2. Diferença em ULP entre as funções padrão e as técnicas implementadas.

A Figura 2 apresenta a diferença em ULP entre a implementação de referência e as versões aproximadas. Em todos os casos, observa-se perda de precisão em relação

²Disponível em: <https://github.com/Victor-Briganti/mathx>

à `glibc`. No entanto, essa métrica deve ser interpretada com cautela, pois uma grande diferença em ULP não implica necessariamente um erro absoluto elevado, especialmente em aplicações tolerantes a erro. Por exemplo, para a entrada -1.8366345867716292 , a função `cos` da `glibc` retorna -0.26271817445294432 , enquanto a versão avaliada por `Estrin` retorna -0.26271817441422191 . Nesse caso, apesar da diferença em ULP, os valores coincidem em aproximadamente dez casas decimais, o que pode ser suficiente para determinadas aplicações.

De modo geral, as aproximações polinomiais apresentaram o melhor equilíbrio entre desempenho e precisão. Já as versões geométricas foram mais rápidas, porém introduziram maior perda de acurácia, chegando a cerca de duas casas decimais de precisão em alguns casos. Assim, a escolha da técnica depende do nível de erro aceitável para cada aplicação.

4. Conclusão

Este trabalho avaliou diferentes implementações aproximadas de funções trigonométricas em ponto flutuante. Os resultados mostram que a `MathX` pode reduzir o tempo de execução em relação às implementações tradicionais, sendo mais adequada para aplicações tolerantes a pequenos erros numéricos. As aproximações polinomiais apresentaram melhor equilíbrio entre desempenho e precisão, enquanto as geométricas obtiveram maior ganho de desempenho com maior perda de acurácia. Como trabalhos futuros, pretende-se estender a avaliação para outras funções matemáticas e investigar novas estratégias de aproximação que reduzam o erro sem comprometer o desempenho.

Referências

- Aanjaneya, M. and Nagarakatte, S. (2023). Fast polynomial evaluation for correctly rounded elementary functions using the RLIBM approach. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, pages 95–107.
- Advanced Micro Devices, Inc. (2026). AOCL-LibM (AMD Math Library). <https://www.amd.com/en/developer/aocl/libm.html>. Accessed: 2026-06-16.
- Briggs, I. and Panchekha, P. (2022). Choosing mathematical function implementations for speed and accuracy. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 522–535.
- Chevillard, S., Joldeş, M., and Lauter, C. (2010). Sollya: An environment for the development of numerical codes. In Fukuda, K., van der Hoeven, J., Joswig, M., and Takayama, N., editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 28–31, Heidelberg, Germany. Springer.
- Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1):5–48.
- Google (2013). Google Benchmark: A microbenchmark support library. <https://github.com/google/benchmark>. Accessed: 2026-06-18.
- Sibidanov, A., Zimmermann, P., and Glondou, S. (2022). The core-math project. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, pages 26–34.