The Journal of Open Source Software

FastTanhSinhQuadrature.jl: High-performance Tanh-Sinh numerical integration in Julia

Stamatis Vretinaris [1]

1 Institute for Mathematics, Astrophysics and Particle Physics, Radboud University, Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands **2** Albert-Einstein-Institut, Max-Planck-Institut für Gravitationsphysik, Callinstraße 38, 30167 Hannover, Germany **3** Leibniz Universität Hannover, 30167 Hannover, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @mongibellili
- @ranocha

Submitted: 29 January 2026
Published: unpublished

Summary

Numerical integration is a cornerstone of scientific computing, essential for evaluating integrals that cannot be solved analytically. The Tanh-Sinh (or Double Exponential) quadrature, originally proposed by Takahasi & Mori (1973), is a powerful technique known for its high accuracy and efficiency, particularly for integrands with endpoint singularities. `FastTanhSinhQuadrature.jl` provides a high-performance, arbitrary-precision implementation of this method in Julia. It leverages modern compiler technologies to achieve significant speedups over traditional implementations while maintaining rigorous mathematical precision.

Statement of Need

In many fields of physics and engineering, researchers encounter integrals with singularities at the boundaries. Standard Gaussian quadrature rules often fail or require an excessive number of points to converge in these cases. While other integration libraries exist, they typically lack a combination of robustness, performance, and flexibility. `FastTanhSinhQuadrature.jl` addresses these needs by:

1. **Robustness**: Automatically handling endpoint singularities without manual coordinate transformations.
2. **Performance**: Utilizing SIMD (Single Instruction, Multiple Data) instructions for rapid evaluation.
3. **Flexibility**: Supporting arbitrary precision types (e.g., `BigFloat`) and multidimensional integration.

This package implements a rigorous Tanh-Sinh scheme with an optimized "window selection" strategy enabling SIMD-accelerated execution paths, making it ideal for large-scale simulations where both speed and precision are critical.

State of the field

Numerical integration is a well-established field, with mature implementations in libraries such as **Boost** (Boost C++ Libraries, 2024) (C++), **SciPy** (Virtanen et al., 2020) (Python), and **mpmath** (Johansson & others, 2023) (Python). Within the Julia ecosystem, packages like `QuadGK.jl` (Johnson, 2013) and `HCubature.jl` (Johnson, 2017) provide detailed adaptive Gauss-Kronrod and h-adaptive cubature methods. However, high-performance implementations specifically of the **Tanh-Sinh quadrature** are less common.

The Journal of Open Source Software

Most existing Tanh-Sinh implementations rely on dynamic convergence checks within the summation loop. This introduces conditional branching that prevents modern compilers from applying SIMD vectorization, restricting solvers to scalar execution speeds. `FastTanhSinhQuadrature.jl` overcomes this by adopting the "window selection" strategy described by Vanherck et al. (2020). By analytically pre-calculating the optimal step size h and truncation index n based on floating-point precision, the algorithm eliminates runtime checks, creating a branch-free inner loop amenable to optimization by `LoopVectorization.jl` (Elrod, n.d.).

Mathematics

The Tanh-Sinh quadrature computes integrals of the form $I = \int_{-1}^{1} f(x)\,\mathrm{d}x$ by applying the variable transformation $x = \tanh(\frac{\pi}{2}\sinh(t))$ proposed by Takahasi & Mori (1973). This maps the finite interval to the real line, where the integrand decays double-exponentially. The integral is then approximated using the trapezoidal rule over the infinite domain, truncated to a finite window $[-n, n]$.

For a detailed derivation of the quadrature weights, error bounds, and the window selection strategy used to determine h and n, the reader is referred to Takahasi & Mori (1973) and Vanherck et al. (2020).

Software Design

The package balances ease of use with maximum performance through a two-tier API:

1. **High-Level API** (`quad`): A drop-in replacement for standard quadrature functions, handling adaptivity, singularities, and infinite domains automatically.
2. **Low-Level API** (`integrateND_avx`): Allows users to pre-compute quadrature nodes and weights for reuse across millions of integrals, eliminating allocation overhead in tight loops.

Key implementation features include:

- **Window Selection**: Uses the method of Vanherck et al. (2020) to pre-determine integration bounds, enabling branch-free loops.
- **SIMD Optimization**: Leverages `LoopVectorization.jl` to vectorize evaluation loops, yielding 2-3x speedups over scalar codes.
- **Static Allocation**: For moderate node counts, weights and nodes can be stored in `StaticArrays`, eliminating heap allocations.
- **Arbitrary Precision**: Supports generic number types (`BigFloat`, `Double64`) by dynamically deriving quadrature parameters from machine epsilon.

Research Impact

`FastTanhSinhQuadrature.jl` has been integrated as a backend for `Integrals.jl` (Widmann & Rackauckas, 2020), ensuring widespread availability within the SciML ecosystem.

Performance

Benchmarks against `FastGaussQuadrature.jl` (Townsend et al., 2013) demonstrate that our SIMD-optimized Tanh-Sinh implementation (`integrate1D_avx`) achieves competitive or superior performance. For singular integrands like $\sqrt{1-x^2}$, we observe speedups of approximately **2.4x**. For generic smooth functions like e^x, the solver is approximately **2x** faster than standard Gaussian quadrature.

Detailed performance benchmarks, timing tables, and convergence plots are available in the software repository.

Usage

Installation

```julia
using Pkg
Pkg.add("FastTanhSinhQuadrature")
```

Basic Integration

```julia
using FastTanhSinhQuadrature

# Integrate exp(x) from 0 to 1
val = quad(exp, 0.0, 1.0) # ≈ 1.71828...

# Handle singularities: 1/sqrt(x)
val = quad(x -> 1/sqrt(x), 0.0, 1.0) # ≈ 2.0
```

High-Performance Pre-computation

```julia
# Pre-compute nodes/weights for Float64
x, w, h = tanhsinh(Float64, Val(80))

# Reuse in tight loops (zero-allocation)
f(t) = sin(t)^2
integral = integrate1D_avx(f, 0.0, π, x, w, h)
```

Convergence

Convergence tests for various integrands are shown below. The method exhibits rapid exponential convergence characteristic of the Tanh-Sinh scheme.

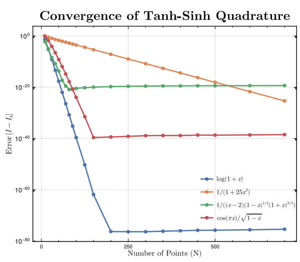

Figure 1: Convergence of Tanh-Sinh Quadrature compared to other methods.

AI usage disclosure

During the development of this package, the author utilized Gemini (Google) for assistance with documentation, debugging and the generation of the first draft of this paper. The author has reviewed and edited all AI-generated content to ensure accuracy and adherence to the package's coding standards.

Acknowledgements

The author acknowledges the developers of `LoopVectorization.jl` for providing the tools that enabled the performance optimizations in this package.

References

Boost C++ Libraries. (2024). *Boost c++ libraries*. https://www.boost.org/

Elrod, C. (n.d.). *LoopVectorization.jl*. https://github.com/JuliaSIMD/LoopVectorization.jl

Johansson, F., & others. (2023). *Mpmath: A Python library for arbitrary-precision floating-point arithmetic*. http://mpmath.org/

Johnson, S. G. (2013). *QuadGK.jl: Gauss-kronrod integration in julia*. https://github.com/JuliaMath/QuadGK.jl

Johnson, S. G. (2017). *HCubature.jl: H-adaptive multidimensional integration in julia*. https://github.com/JuliaMath/HCubature.jl

Takahasi, H., & Mori, M. (1973). Double exponential formulas for numerical integration. *Publications of the Research Institute for Mathematical Sciences*, *9*(3), 721–741. https://doi.org/10.2977/PRIMS/1195192451

Townsend, A., Hale, N., & Olver, S. (2013). *FastGaussQuadrature.jl*. https://github.com/JuliaApproximation/FastGaussQuadrature.jl

Vanherck, J., Sorée, B., & Magnus, W. (2020). Tanh-sinh quadrature for single and multiple integration using floating-point arithmetic. *arXiv Preprint arXiv:2007.15057*. https://doi.org/10.48550/arXiv.2007.15057

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Widmann, D., & Rackauckas, C. (2020). *Integrals.jl: A common interface for numerical integration in julia*. https://github.com/SciML/Integrals.jl