The Journal of Open Source Software

pyaudi: A truncated Taylor polynomial algebra toolbox for differentiable intelligence, automatic differentiation, and verified integration applications.

Dario Izzo [1], **Francesco Biscani** [2], **and Sean Cowan** [1]

1 Advanced Concepts Team, European Space Research and Technology Center **2** Skylon Dynamics

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Olexandr Konovalov

Reviewers:

- @tesch1
- @scarlehoff

Submitted: 09 October 2025
Published: unpublished

Summary

pyaudi is an open-source Python toolbox developed at the European Space Agency that provides high-order forward-mode automatic differentiation in a multivariate setting. Built on C++ class templates and exposed to Python via pybind11, it implements the algebra of truncated Taylor polynomials. This design allows its generalized dual number type to serve as a drop-in replacement for scalar types like floats, with operator overloading ensuring automatic derivative propagation. The underlying C++ codebase `audi` can also be used directly, offering greater flexibility for instantiating the algebra over different fields.

All standard mathematical operations are supported, leveraging the nilpotency of exponentiation in truncated Taylor polynomial algebra. Within a truncated algebra $\mathcal{A}^n$, any polynomial $p \in \mathcal{A}^n$ with vanishing constant term evaluates to zero when raised to a power greater than the truncation order $(m > n)$. This property enables efficient and exact computation of derivatives to arbitrary order while preserving the simplicity of standard numerical code.

Beyond truncated Taylor polynomials, `pyaudi` also implements Taylor models (Makino, 1998), which combine these polynomials with an interval bounding their truncation error, along with miscellaneous algorithms for applications in differential intelligence, high-order automatic differentiation, verified integration, and related fields.

Statement of need

`pyaudi` enables users to compute and manipulate order-n Taylor expansions of generic computational graphs, while providing rigorous bounds on truncation errors through associated Taylor models. These Taylor representations of program outputs can be used for fast Monte Carlo simulations, rigorous uncertainty analysis, local inversion of output–input relations, and high-order sensitivity studies. The package implements the high-order automatic differentiation methodology of Berz and Makino ((Berz et al., 2014), (Makino, 1998)), introducing novel implementation details in polynomial multiplication routines and Taylor model bounding.

Existing libraries with similar capabilities

At the time of writing, two main open-source libraries perform computations similar to those in `pyaudi`. The first is the C/C++ library DACE (Massari et al., 2018), which implements the full differential algebra of truncated Taylor polynomials with float coefficients. Unlike `pyaudi`, DACE uses a memory-intensive polynomial multiplication routine for storing monomial coefficients. As shown in the separate paper_results.md file (in the paper directory of the paper branch of the repository), this gives DACE an advantage for single evaluations at low orders, though the benefit decreases for batched computations and higher orders.

The second project comprises the Julia libraries TaylorSeries.jl and TaylorModels.jl (Benet et al., 2019), which implement Taylor models to compute rigorous bounds on Taylor series. Their approach, however, differs substantially from `pyaudi`, and preliminary results — also presented in the separate paper_results.md file — indicate that `pyaudi` significantly outperforms them in the tested cases.

In recent years, numerous automatic differentiation toolboxes have emerged for machine learning and optimization, notably JAX (Bradbury et al., 2018), TensorFlow (Abadi et al., 2015), and PyTorch (Paszke et al., 2019) (autograd). These excel at low-order derivatives (first and second), where reverse-mode differentiation is highly efficient. For higher-order derivatives, however, most implementations perform poorly; while experimental features such as JAX's jet module (Bettencourt et al., 2019) show promise, they remain immature.

Key aspects

The main features of `pyaudi` are:

- **Truncated polynomial algebra**, powered by Obake, a C++ library for symbolic manipulation of sparse multivariate polynomials, truncated power series, and Poisson series. Unlike other packages, which often suffer from severe memory bottlenecks as the polynomial order or the number of variables increases, `pyaudi` avoids large static memory allocations by adopting a sparse, dynamic approach. This remains memory-efficient at the cost of additional bookkeeping, where sparse polynomials are the area of greatest benefit. The use of templates allows to instantiate the algebra over different fields such as floats, quadruple precision floats, vectorized floats, etc..
- **Vectorized coefficients**, enabling the simultaneous evaluation of identical computational graphs at multiple expansion points. This feature makes it possible to compute high-order derivatives on multiple points, while alleviating the overhead introduced by the sparse bookkeeping of Obake.
- **Taylor models with Bernstein polynomial bounding**, used to enclose the range of multivariate polynomials.
- **Map inversion algorithm**, implementing the method described in (Berz et al., 2014). This feature enables the local inversion of input–output relations arising in generic computational graphs.

Ongoing research

(Acciarini et al., 2024) extended the principle of numerical continuation to find solutions in non-linear dynamical systems in the space domain to the moments of a probability density function, using `pyaudi` to calculate the high-order derivatives in the Circular Restricted Three-Body Problem (CR3BP). Similarly, `pyaudi` was used in (Acciarini et al., 2025) for various test cases involving non-Gaussian uncertainty distributions to calculate the high-order derivatives. (Caleb & Lizy-Destrez, 2020) did work on exploring Differential Algebra (DA) for accelerating Monte Carlo simulations, which is enabled by libraries such as `pyaudi`. For on-board applications, `pyaudi` was used in (Burnett & Topputo, 2025) to implement convex guidance algorithms by describing the solutions using DA. (Izzo et al., 2017) explored Cartesian Genetic Programming and made it differentiable with high-order automatic differentiation, enabled by `pyaudi`.

Acknowledgement of financial support

No financial support was provided for the development of this software.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Acciarini, G., Baresi, N., Lloyd, D. J., & Izzo, D. (2024). Stochastic continuation of trajectories in the circular restricted three-body problem via differential algebra. *arXiv Preprint arXiv:2405.18909*.

Acciarini, G., Baresi, N., Lloyd, D. J., & Izzo, D. (2025). Nonlinear propagation of non-gaussian uncertainties. *Journal of Guidance, Control, and Dynamics*, *48*(4), 903–913.

Benet, L., Forets, M., Sanders, D., & Schilling, C. (2019). TaylorModels. Jl: Taylor models in julia and their application to validated solutions of ODEs. In *SWIM* (pp. 15–16).

Berz, M., Makino, K., & Wan, W. (2014). *An introduction to beam physics*. Taylor & Francis.

Bettencourt, J., Johnson, M. J., & Duvenaud, D. (2019). Taylor-mode automatic differentiation for higher-order derivatives in JAX. *Program Transformations for ML Workshop at NeurIPS 2019*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Burnett, E. R., & Topputo, F. (2025). Rapid nonlinear convex guidance using a monomial method. *Journal of Guidance, Control, and Dynamics*, *48*(4), 736–756. https://doi.org/10.2514/1.G008512

Caleb, T., & Lizy-Destrez, S. (2020). Can uncertainty propagation solve the mysterious case of snoopy? *International Conference on Uncertainty Quantification & Optimisation*, 109–128. https://doi.org/10.1007/978-3-030-80542-5_8

Izzo, D., Biscani, F., & Mereta, A. (2017). Differentiable genetic programming. *European Conference on Genetic Programming*, 35–51. https://doi.org/10.1007/978-3-319-55696-3

Makino, K. (1998). *Rigorous analysis of nonlinear motion in particle accelerators* [PhD thesis]. Michigan State University.

Massari, M., Di Lizia, P., Cavenago, F., & Wittig, A. (2018). Differential algebra software library with automatic code generation for space embedded applications. In *2018 AIAA information systems-AIAA infotech@ aerospace* (p. 0398). https://doi.org/10.2514/6.2018-0398

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., & others. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.