The Journal of Open Source Software

pyaudi: A 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 (Noordwijk, NL) **2** Max Planck Institute for Astronomy (Heidelberg, DE)

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

`pyaudi` is a Python toolbox developed at the European Space Agency that implements the algebra of Taylor truncated polynomials to achieve any order, forward mode, automatic differentiation in a multivarate setting. The forward mode automatic differentiation is implemented via C++ class templates exposed to python using pybind11. This allows the generalized dual number type to behave like a drop-in replacement for floats (or other scalar types), while operator overloading propagates derivatives automatically.

On top of the algebra of Taylor truncated polynomials, `pyaudi` offers an implementation of Taylor models, which combine truncated Taylor polynomials with an interval bounding its truncation error as well as a number of miscellaneous algorithms useful for applications in differential intelligence, automatic differentiation, verified integration and more.

Statement of need

`pyaudi` enables researchers to compute and manipulate order n Taylor expansions of generic computational trees as well as bound precisely the truncation error introduced using its corresponding Taylor model. The resulting polynomial representations of the program outputs can be used to perform fast Monte Carlo simulations, rigorous uncertainty analyses local inversions output-input relations.

Key aspects

The key novel aspects of the `pyaudi` package are:

- Truncated polynomial arithmetic to arbitrary dimensions using Obake a C++ computer algebra library for the symbolic manipulation of sparse multivariate polynomials and other closely-related symbolic objects such as truncated power series and Poisson series. Packages implementing similar functionalities tend to encounter memory issues as the order and number of variables increase, `pyaudi` keeps the problem manageable memory wise as it avoids allocation of huge static memory arays.
- A vectorized version of the generalized dual numbers allowing to evaluate simultaneously the same computatinal graphs (and thus high order tensors) at different expansion ponts, thus gradually cancelling the overhead of managing bookeeping along the computational graph.
- Effective Taylor models for verified integration of ODEs implemented using Bernstein polynomials for bounding the range of multivariate polynomials. The only other open-source

The Journal of Open Source Software

package, called TaylorModels.jl, calculates bounds using Horner's scheme combined with interval arithmetic. A quick test in the next section shows significantl speedup for even a relatively simple trivariate polynomial.

- An implementation of the map inversion algorithm allowing to invert (locally) the input-output relation of generic computational trees.

Comparison with TaylorModels.jl

To quickly test the performance of the Taylor model implementation in `pyaudi` against TaylorModels.jl, we take three functions; one univariate, one bivariate and one trivariate shown below. We then construct Taylor models of all the variables separately and time the evaluation of the function value as a Taylor model.

$$
\begin{aligned}
f(x,y,z) = {} & \frac{4\tan(3y)}{3x + x\sqrt{\frac{6x}{-7(x-8)}}} - 120 - 2x - 7z(1+2y) \\
& - \sinh\left(0.5 + \frac{6y}{8y+7}\right) + \frac{(3y+13)^2}{3z} - 20z(2z-5) \\
& + \frac{5x\tanh(0.9z)}{\sqrt{5y}} - 20y\sin(3z)
\end{aligned}
$$

$$
g(x,y) = \sin(1.7x + 0.5)(y+2)\sin(1.5y)
$$

$$
h(x) = x(x-1.1)(x+2)(x+2.2)(x+2.5)(x+3)\sin(1.7x+0.5)
$$

Dimension	Package	Remainder Bound (Order 1)	Remainder Bound (Order 15)	Speed Comparison
h(x)	TaylorModels.jl	1e-15	1e-15	~1.5–2× faster than pyaudi
	pyaudi	1e+2	1e-5	~1.5–2× slower than TaylorModels.jl
g(x, y)	TaylorModels.jl	1e+1	1e-6	Slower: pyaudi is 5× faster (order 3), 15× faster (order 15), 7800× faster (order 1, edge case)
	pyaudi	1e+1	1e-6	Faster (see above)
f(x, y, z)	TaylorModels.jl	1e+0	1e-11	Slower: pyaudi is 8× faster (order 3), 155× faster (order 15), 13000× faster (order 1, edge case)
	pyaudi	1e-1	1e-17	Faster (see above)

In the table above, a clear trend can be seen both in terms of speed and accuracy. For univariate Taylor models, TaylorModels.jl is marginally faster and numerically precise. At two dimensions, the remainder bounds are of equal size, but `pyaudi` is significantly faster, with the speedup increasing with the order of the polynomial. At three dimensions, `pyaudi` produces significantly tighter bounds and is again significantly faster, with the speedup increasing with the order of the polynomial.

References

A number of references to relevant work and algorithms implemented in `pyaudi` are:

- (Biscani, 2020)
- (Makino, 1998)
- (Titi & Garloff, 2019)

Other software packages that do similar things are:

- JAX (Bradbury et al., 2018)
- TensorFlow (Abadi et al., 2015)
- PyTorch (Paszke et al., 2019)
- COSY INFINITY (Makino & Berz, 2006)
- DACE (Massari et al., 2018)
- TaylorSeries.jl/TaylorModels.jl (Benet et al., 2019)
- CORA (Althoff, 2015)

Ongoing research

- EclipseNET (Acciarini, Biscani, et al., 2024) (Acciarini, Izzo, et al., 2025)
- CR3BP stochastic continuation (Acciarini, Baresi, et al., 2024)
- Long-term propagation (Caleb & Lizy-Destrez, 2020)
- dSGP4 (Acciarini, Baydin, et al., 2025)

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., Baydin, A. G., & Izzo, D. (2025). Closing the gap between SGP4 and high-precision propagation via differentiable programming. *Acta Astronautica*, *226*, 694–701.

Acciarini, G., Biscani, F., & Izzo, D. (2024). EclipseNETs: A differentiable description of irregular eclipse conditions. *arXiv Preprint arXiv:2408.05387*.

Acciarini, G., Izzo, D., & Biscani, F. (2025). EclipseNETs: Learning irregular small celestial body silhouettes. *arXiv Preprint arXiv:2504.04455*.

Althoff, M. (2015). An introduction to CORA 2015. *Proc. Of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, 120–151. https://doi.org/10.29007/zbkv

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).

Biscani, F. (2020). *Obake: A c++17 library for the symbolic manipulation of sparse polynomials & co.* (Version 0.9.0). https://github.com/bluescarni/obake

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

Caleb, T., & Lizy-Destrez, S. (2020). Can uncertainty propagation solve the mysterious case of snoopy? *International Conference on Uncertainty Quantification & Optimisation*, 109–128.

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

Makino, K., & Berz, M. (2006). Cosy infinity version 9. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, *558*(1), 346–350.

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).

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*.

Titi, J., & Garloff, J. (2019). Matrix methods for the tensorial bernstein form. *Applied Mathematics and Computation*, *346*, 254–271.