


smolyax: a high-performance implementation of the Smolyak interpolation operator in JAX

Josephine Westermann ¹¶ and Joshua Chen ²

¹ Heidelberg University, Germany  ² Colorado State University, USA  ¶ Corresponding author

DOI: [10.21105/joss.08505](https://doi.org/10.21105/joss.08505)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Patrick Diehl](#) 

Reviewers:

- [@MoraruMaxim](#)
- [@mjcarley](#)

Submitted: 15 May 2025

Published: 14 August 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

The `smolyax` library provides interpolation capabilities for arbitrary multivariate and vector-valued functions $f : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ for any $d_{\text{in}}, d_{\text{out}} \in \mathbb{N}$. It implements the Smolyak interpolation operator, which is known to overcome the curse-of-dimensionality plaguing naive multivariate interpolation ([Barthelmann et al., 2000](#)) and uses the barycentric interpolation formula ([Berrut & Trefethen, 2004](#)) for numerical stability. The implementation is based on JAX ([Bradbury et al., 2018](#)), a free and open-source Python library for high-performance computing that integrates with the Python and NumPy numerical computing ecosystem. Thanks to JAX's device management, `smolyax` runs natively on both CPU and GPU. While implementing Smolyak interpolation in JAX is challenging due to the highly irregular data structures involved, `smolyax` overcomes this by employing a tailored batching and padding strategy that enables efficient vectorization.

`smolyax` supports interpolation on bounded or unbounded domains via Leja ([M. A. Chkifa, 2013](#)) or Gauss-Hermite ([Abramowitz & Stegun, 1964](#)) node sequences, respectively. It provides efficient Numba-accelerated routines to generate isotropic or anisotropic total degree multi-index sets ([Adcock et al., 2022, sec. 2.3.2](#)), which are the key ingredient to generate the high-dimensional sparse grids ([Bungartz & Griebel, 2004](#)) of interpolation nodes required by the Smolyak interpolation operator. Additional types of node sequences or multi-index sets can be incorporated easily by implementing a minimalistic interface. The `smolyax` interpolant provides further functionality to evaluate its gradient as well as compute its integral.

Statement of Need

Polynomial expansion is a well-studied and powerful tool in applied mathematics, with important applications in surrogate modeling, uncertainty quantification and inverse problems, see e.g., [Adcock et al. \(2022\)](#), [Düng et al. \(2023\)](#), [Zech \(2018\)](#), [A. Chkifa et al. \(2015\)](#), [Herrmann et al. \(2024\)](#), [Westermann et al. \(2025\)](#), and references therein. Smolyak interpolation offers a practical way to construct polynomial approximations with known error bounds for a wide range of function classes, see e.g., [Barthelmann et al. \(2000\)](#), [A. Chkifa et al. \(2015\)](#), and [Adcock et al. \(2022\)](#).

Several libraries provide CPU-based high-dimensional interpolation functionality, for example Chaospy ([Feinberg & Langtangen, 2015](#)), UQLab ([Marelli & Sudret, 2014](#)), The Sparse Grid Matlab Kit ([Piazzola & Tamellini, 2024](#)), PyApprox ([Jakeman, 2023](#)), MUQ ([Parno et al., 2021](#)), and UncertainSCI ([Tate et al., 2023](#)). The GPU support that is necessary in practice to go from moderate to high dimensions is offered so far only by Tasmanian ([Stoyanov, 2015](#)). Benchmark experiments suggest that while asymptotic runtime of the Smolyak interpolator in Tasmanian scale better as the output dimension d_{out} increases, `smolyax` offers competitive asymptotic runtimes for increasing d_{in} and input data set size.

A vectorizable implementation of the Smolyak operator

Recall that given a domain $D \subset \mathbb{R}$ and set of $\nu + 1 \in \mathbb{N}$ pairwise distinct interpolation points $(\xi_i^\nu)_{i=0}^\nu \subset D$, the univariate polynomial interpolation operator I^ν maps a function $f : D \rightarrow \mathbb{R}$ onto the unique polynomial $I^\nu[f] \in \mathbb{P}_\nu := \text{span}\{x^i : i = 0, \dots, \nu\}$ of maximal degree ν such that $f(\xi_i^\nu) = I^\nu[f](\xi_i^\nu)$ for all $i \in \{0, 1, \dots, \nu\}$.

Tensorized interpolation generalizes univariate interpolation to multivariate functions defined on a tensor-product domain $D = \otimes_{j=1}^d D_1$ with $D_1 \subset \mathbb{R}$ and $d \in \mathbb{N}$ by defining $I^\nu := \otimes_{j=1}^d I^{\nu_j}$, where $\nu \in \mathbb{N}_0^d$ is a multi-index characterizing the maximal polynomial degree in each dimension. Since $I^\nu[f] \in \mathbb{P}_\nu := \text{span}\{x^\mu : \mu \leq \nu\}$, this approach suffers from the curse of dimensionality as d increases.

Smolyak interpolation (Barthelmann et al., 2000; Smolyak, 1963) overcomes this issue by introducing polynomial ansatz spaces $\mathbb{P}_\Lambda := \text{span}\{x^\mu : \mu \in \Lambda\}$ parametrized by downward closed multi-index sets $\Lambda \subset \mathbb{N}_0^d$. The resulting interpolation operator is a linear combination of tensorized interpolation operators:

$$I^\Lambda := \sum_{\nu \in \Lambda} \zeta_{\Lambda, \nu} I^\nu, \quad \zeta_{\Lambda, \nu} := \sum_{e \in \{0,1\}^d : \nu + e \in \Lambda} (-1)^{|e|}.$$

Implementing this operator in a vectorized form suitable for high-performance computing is nontrivial, as vectorization requires inputs to conform to a uniform structure. However, each tensorized interpolant in the equation above involves reducing a higher-order tensor of unique shape ν via multiplication with one vector per dimension. A naive strategy would be to zero-pad all tensors in this equation to the smallest common shape $(\max_{\nu \in \Lambda} (\nu_j))_{j=1}^d$. This, however, reintroduces the curse of dimensionality, as memory requirements grow exponentially with d .

smolyax strikes a balance between handling small, uniquely shaped tensors and large, identically shaped ones. The key idea is to group tensors by their number of active dimensions and prepare them for vectorized processing within each group. In particular, this involves

1. Dropping indices ("squeezing") of non-active dimensions, i.e., j with $\nu_j = 0$;
2. Permuting the remaining active dimensions in descending order; and
3. Zero-padding all tensors with the same number of active dimensions to the smallest common shape.

This reorganizes the tensors into a few large, structured blocks enabling fast vectorized processing. Asymptotically, in both dimension and size of the polynomial space, the method requires only a logarithmic-factor increase in overall memory compared with the raw tensors.

Acknowledgements

We thank Thomas O'Leary-Roseberry and Jakob Zech for insightful discussions in the early stages of this project.

References

- Abramowitz, M., & Stegun, I. A. (1964). *Handbook of mathematical functions with formulas, graphs, and mathematical tables: Vols. No. 55* (p. xiv+1046). U. S. Government Printing Office, Washington, DC.
- Adcock, B., Brugiapaglia, S., & Webster, C. G. (2022). *Sparse polynomial approximation of high-dimensional functions* (Vol. 25, p. xvii+292). Society for Industrial; Applied Mathematics (SIAM), Philadelphia, PA. <https://doi.org/10.1137/1.9781611976885>

- Barthelmann, V., Novak, E., & Ritter, K. (2000). High dimensional polynomial interpolation on sparse grids. In *Adv. Comput. Math.* (Vol. 12, pp. 273–288). <https://doi.org/10.1023/A:1018977404843>
- Berrut, J.-P., & Trefethen, L. N. (2004). Barycentric Lagrange interpolation. *SIAM Rev.*, 46(3). <https://doi.org/10.1137/S0036144502417715>
- 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 0.5.1). <http://github.com/jax-ml/jax>
- Bungartz, H.-J., & Griebel, M. (2004). Sparse grids. *Acta Numer.*, 13, 147–269. <https://doi.org/10.1017/S0962492904000182>
- Chkifa, A., Cohen, A., & Schwab, C. (2015). Breaking the curse of dimensionality in sparse polynomial approximation of parametric PDEs. *J. Math. Pures Appl.* (9), 103(2), 400–428. <https://doi.org/10.1016/j.matpur.2014.04.009>
- Chkifa, M. A. (2013). On the Lebesgue constant of Leja sequences for the complex unit disk and of their real projection. *J. Approx. Theory*, 166, 176–200. <https://doi.org/10.1016/j.jat.2012.11.005>
- Düng, D., Nguyen, V. K., Schwab, C., & Zech, J. (2023). *Analyticity and sparsity in uncertainty quantification for PDEs with Gaussian random field inputs* (Vol. 2334, p. xv+205). Springer, Cham. <https://doi.org/10.1007/978-3-031-38384-7>
- Feinberg, J., & Langtangen, H. P. (2015). Chaospy: An open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, 11, 46–57. <https://doi.org/10.1016/j.jocs.2015.08.008>
- Herrmann, L., Schwab, C., & Zech, J. (2024). Neural and spectral operator surrogates: Unified construction and expression rate bounds. *Adv. Comput. Math.*, 50(4), Paper No. 72, 43. <https://doi.org/10.1007/s10444-024-10171-2>
- Jakeman, J. D. (2023). PyApprox: A software package for sensitivity analysis, bayesian inference, optimal experimental design, and multi-fidelity uncertainty quantification and surrogate modeling. *Environmental Modelling & Software*, 170, 105825. <https://doi.org/10.1016/j.envsoft.2023.105825>
- Marelli, S., & Sudret, B. (2014). UQLab: A framework for uncertainty quantification in matlab. In *Vulnerability, uncertainty, and risk: Quantification, mitigation, and management* (pp. 2554–2563). <https://doi.org/10.1061/9780784413609.257>
- Parno, M., Davis, A., & Seelinger, L. (2021). MUQ: The MIT uncertainty quantification library. *Journal of Open Source Software*, 6(68), 3076. <https://doi.org/10.21105/joss.03076>
- Piazzola, C., & Tamellini, L. (2024). Algorithm 1040: The Sparse Grids Matlab Kit – a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification. *ACM Transactions on Mathematical Software*, 50(1). <https://doi.org/10.1145/3630023>
- Smolyak, S. A. (1963). Quadrature and interpolation formulas for tensor products of certain classes of functions. *Doklady Akademii Nauk*, 148, 1042–1045.
- Stoyanov, M. (2015). *User manual: TASMANIAN sparse grids* (ORNL/TM-2015/596). Oak Ridge National Laboratory.
- Tate, J., Liu, Z., Bergquist, J. A., Rampersad, S., White, D., Charlebois, C., Rupp, L., Brooks, D. H., MacLeod, R. S., & Narayan, A. (2023). UncertainSCI: A Python package for noninvasive parametric uncertainty quantification of simulation pipelines. *Journal of Open Source Software*, 8(90), 4249. <https://doi.org/10.21105/joss.04249>

Westermann, J., Huber, B., O'Leary-Roseberry, T., & Zech, J. (2025). *Performance of neural and polynomial operator surrogates*. Manuscript in preparation.

Zech, J. (2018). *Sparse-Grid Approximation of High-Dimensional Parametric PDEs*. <https://doi.org/10.3929/ethz-b-000340651>