Minterpy: multivariate polynomial interpolation in Python

Damar Wicaksono [1], **Uwe Hernandez Acosta** [1], **Sachin Krishnan Thekke Veettil** [3], **Jannik Kissinger** [3,4], **and Michael Hecht** [1,2]

1 Center for Advanced Systems Understanding (CASUS) - Helmholtz-Zentrum Dresden-Rossendorf (HZDR), Germany **2** University of Wrocław, Poland **3** Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany **4** Technische Universität Dresden, Germany

Summary

Interpolation is essential in various computational tasks, including function approximation, curve fitting, numerical integration, differential geometry, spectral methods, optimization, and uncertainty quantification.

Minterpy is an open-source Python package designed for multivariate polynomial interpolation. It provides stable and accurate interpolating polynomials for approximating a wide range of functions. Key features include:

- Polynomial interpolation on properly selected nodes and regression on arbitrary nodes.
- Differentiation and integration operations on the polynomials.
- Addition, subtraction, and multiplication operations on the polynomials.

Minterpy's long-term vision is to provide researchers and engineers a software solution that mitigates the curse of dimensionality commonly associated with interpolation tasks.

Statement of need

As a means of approximating functions, global polynomials—where a single polynomial is defined over the entire domain—offer several advantages. For sufficiently smooth functions, global polynomials can achieve high accuracy with a smaller number of data points (sampled over the entire domain) compared to local piecewise polynomials. Additionally, their relatively simple structure facilitates many common numerical operations. These operations include differentiation, integration, addition, subtraction, and multiplication (Trefethen, 2019).

The Stone-Weierstrass theorem establishes that any continuous function on a bounded domain in multiple dimensions can be approximated uniformly to arbitrary precision by multivariate global polynomials (de Branges, 1959). However, the theorem does not specify a concrete method for constructing such approximating polynomials. Various techniques can be employed to build approximating polynomials, such as least square approximations. Minterpy focuses on constructing approximating global polynomials using one of the earliest and most established methods: interpolation (Goldstine, 1977).

Polynomial interpolation is based on the principle that, in one dimension, there exists a unique polynomial $Q_{f,n}$ of degree n that interpolates a function $f : \Omega \rightarrow \mathbb{R}$ in a bounded domain Ω with $n+1$ *distinct (unisolvent*[1]*) interpolation nodes (or points)* P_n such that

$$Q_{f,n}(p_i) = f(p_i), \ \forall p_i \in P_n \subset \Omega, \ i = 0, \dots, n. \tag{1}$$

---

[1]Unisolvent here means that the interpolating polynomial can be uniquely determined by the given interpolation nodes. In one dimension, this implies that the nodes are of distinct values.

Polynomial interpolation has its roots in the works of Newton, Euler, Lagrange, and others (Meijering, 2002), and its significance in mathematics and computing is well-established (Cools, 2002; Xiu, 2009).

Despite their aforementioned advantages as global polynomials, global interpolating polynomials have a controversial reputation due to several misconceptions (Trefethen, 2011, 2016, 2017a):

- They are often thought to be prone to Runge's phenomenon, whereby increasing the degree of interpolating polynomials worsens the approximation quality.
- Their evaluation is frequently believed to be numerically unstable and susceptible to round-off errors.
- Their extension to multiple dimensions is seen as severely limited by the curse of dimensionality, particularly when using tensor product constructions, which causes the required number of interpolation nodes (i.e., data points) to grow exponentially with the number of spatial dimensions.
- They are said to generally fail to converge to the approximated function as the degree increases, with Faber's theorem often cited to justify this assertion. This view has contributed to a more generally pessimistic outlook on the use of interpolating polynomials for function approximation.

Minterpy addresses these issues by:

- Constructing multivariate interpolating polynomials using appropriate interpolation nodes (e.g., Chebyshev-Lobatto nodes) to help mitigate Runge's phenomenon[2];
- Representing the interpolating polynomials in the Newton basis, combined with Leja ordering of the interpolation nodes to ensure stable evaluation (Breuß et al., 2018; Reichel, 1990; Tal-Ezer, 1991);
- Using a multi-index set to represent the multivariate polynomials, which can be tailored to mitigate the curse of dimensionality while preserving the approximation power of the interpolating polynomials (more on this in the next section).

While Faber's theorem shows that no *interpolating polynomial* can converge for *all* continuous functions, it has been demonstrated that if the function is reasonably smooth, the interpolating polynomials do converge at high algebraic rates for common regular (Lipschitz continuous[3]) functions and at geometric rates for analytic functions (Trefethen, 2017b).

Minterpy shares similar objectives and functionality with Chebfun (Driscoll et al., 2014), a popular MATLAB package[4] designed for numerical computations using interpolating polynomials, specifically Chebyshev polynomials. Chebfun provides features such as root finding, differentiation, and integration for function approximation in up to three dimensions. In contrast, Minterpy supports higher dimensions but with fewer features.

Several Python packages, such as Chaospy (Feinberg & Langtangen, 2015), equadratures (Seshadri & Parks, 2017), PyGPC (Weise et al., 2020), PyThia (Hegemann & Heidenreich, 2023), and UncertainSci (Tate et al., 2023), provide polynomial-based function approximations, primarily for uncertainty quantification (UQ) using generalized polynomial chaos expansion (Xiu & Karniadakis, 2002). These tools frame problems as UQ tasks, where inputs are modeled probabilistically. With few exceptions (notably Chaospy), the resulting polynomials are primarily used for function approximations, accompanied by additional post-processing utilities tailored to UQ tasks (e.g., uncertainty propagation, sensitivity analysis). In contrast, Minterpy offers a simpler, UQ-free approach to function approximation using interpolating polynomials, with fewer barriers to entry, and includes several mathematical operations on the polynomials.

Several other Python packages construct polynomial approximations from data. SciPy (Virtanen

---

[2]Runge's phenomenon arises when using equispaced interpolation nodes, causing large oscillations, especially near the interval's endpoints. Adding more points does not resolve these oscillations.

[3]that is, $|f(x) - f(y)| \leq L|x - y|$ for some constant L and for all $x, y \in \Omega$ where Ω is a bounded domain.

[4]Packages in other languages based on or similar to Chebfun include ApproxFun (Olver et al., 2023) (Julia), and ChebPy (Richardson & others, 2024) and pychebfun (Swierczewski & Verdier, 2024) (Python).

et al., 2020) provides multivariate interpolation methods (e.g., linear, nearest, pchip[5]) for rectilinear grids. ndsplines (Margolis & Lyons, 2019) efficiently implements tensor-product multivariate B-splines that can be differentiated and anti-differentiated. Unlike Minterpy, these tools rely on piecewise local polynomials and are tailored for input/output data pairs. Familiar and widely used, piecewise polynomials—especially splines—remain established tools for polynomial interpolation tasks.

In summary, while not a universal tool for all function approximation problems, Minterpy offers a robust solution for approximating a wide range of multidimensional Lipschitz continuous functions using accurate and stable polynomials. Once obtained, these polynomials can be readily manipulated using standard arithmetic operations, such as addition and multiplication, as well as calculus operations, like differentiation and integration. The significance of this capability extends beyond function approximation, as many numerical methods (e.g., root finding, optimization) can be boiled down to these fundamental operations on functions. By leveraging Minterpy's polynomials, users can conveniently carry out symbolic-like computations that would normally require direct manipulation of function values.

Package overview

Consider an m-dimensional function $f : \mathbf{\Omega} \subset \mathbb{R}^m \to \mathbb{R}$, defined on a hypercube. Minterpy interpolates the function using a polynomial expansion in the Lagrange basis

$$f(\boldsymbol{x}) \approx Q(\boldsymbol{x}) = \sum_{\alpha \in A} f(\boldsymbol{p}_\alpha)\, L_\alpha(\boldsymbol{x}), \tag{2}$$

where $A \subseteq \mathbb{N}^m$, L_α, and $\boldsymbol{p}_\alpha$ are the multi-index set, the Lagrange basis polynomial, and the unisolvent node that correspond to the index element $\boldsymbol{\alpha}$, respectively. The set $\{\boldsymbol{p}_\alpha\}_{\alpha \in A}$ forms the interpolation grid.

Each basis polynomial satisfies the Kronecker delta condition

$$L_\alpha(p_\beta) = \delta_{\alpha,\beta}, \;\; p_\beta \in \{\boldsymbol{p}_\alpha\}_{\alpha \in A}, \;\; \boldsymbol{\alpha} \in A,$$

ensuring $Q(\boldsymbol{x})$ in Equation 2 is an interpolating polynomial.

The multi-index set A determines polynomial coefficients, unisolvent nodes, and function evaluations. In Minterpy, the default is a downward-closed set $A_{m,n,p}$ with spatial dimension $m \in \mathbb{N}_{>0}$, polynomial degree $n \in \mathbb{N}$, and ℓ_p-degree $p \in \mathbb{R}_{>0}$. The set is defined as

$$A_{m,n,p} = \{\boldsymbol{\alpha} \in \mathbb{N}^m : \|\boldsymbol{\alpha}\|_p = (\alpha_1^p + \cdots + \alpha_m^p)^{1/p} \leq n\}.$$

Here, typical choices for p are 1.0, 2.0, and ∞, representing the total, Euclidean, and maximum degree (tensor-product), respectively. These values for p correspond to polynomial, sub-exponential, and exponential growth of the set size as a function of the spatial dimension. Consequently, the maximum degree set faces a severe curse of dimensionality due to the rapid growth of the set size.

It has been shown that the Euclidean degree $p = 2.0$ offers the best compromise for isotropic functions (where each variable has the same importance)[6], as its convergence rate matches that of $p = \infty$ with respect to the polynomial degree, yet with a significantly smaller multi-index set. In contrast, while the size of the multi-index set for $p = 2.0$ is larger than that for $p = 1.0$, the gain in accuracy more than compensates for the increased cost (Hecht et al., 2025; Trefethen, 2017b).

---

[5]Piecewise cubic Hermite interpolating polynomial.

[6]Incorporating anisotropy (e.g., via an adaptive scheme) enables sparser polynomials. While Minterpy does not yet support adaptivity, users can define custom downward-closed multi-index sets for interpolation.

Deriving multidimensional Lagrange bases for non-tensorial grids is challenging. Minterpy uses the Newton basis for efficient evaluation and differentiation

$$Q(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha} \in A} c_{\boldsymbol{\alpha}} \, N_{\boldsymbol{\alpha}}(\boldsymbol{x}), \;\; N_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \prod_{i=1}^{m} \prod_{j=0}^{\alpha_i - 1} (x_i - q_j), \;\; q_j \in P_i,$$

where $c_{\boldsymbol{\alpha}}$ and $N_{\boldsymbol{\alpha}}$ are the Newton coefficient and Newton polynomial that correspond to the index element $\boldsymbol{\alpha}$, respectively; P_i is a set of interpolation nodes in each dimension. Using Leja-ordered Chebyshev-Lobatto interpolation nodes by default, Newton basis offers numerical stability (Breuß et al., 2018; Reichel, 1990; Tal-Ezer, 1991). Computing Newton coefficients, based on the Lagrange coefficients and interpolation grid, via a multidimensional divided-difference scheme (DDS) is a key step in Minterpy (Hecht et al., 2025).

Minterpy also supports other polynomial bases, including the canonical (monomial) and Chebyshev (first kind) bases, along with transformations between them.

Minterpy polynomials for function approximation

Minterpy prioritizes stable and accurate function approximation through polynomial interpolations, even for high-degree polynomials. Consider, the Runge function:

$$f(\boldsymbol{x}) = \frac{1}{1 + \|\boldsymbol{x}\|^2}, \;\; \boldsymbol{x} \in [-1, 1]^m,$$

commonly used to demonstrate Runge's phenomenon, a pitfall in high-degree interpolation with equispaced points.
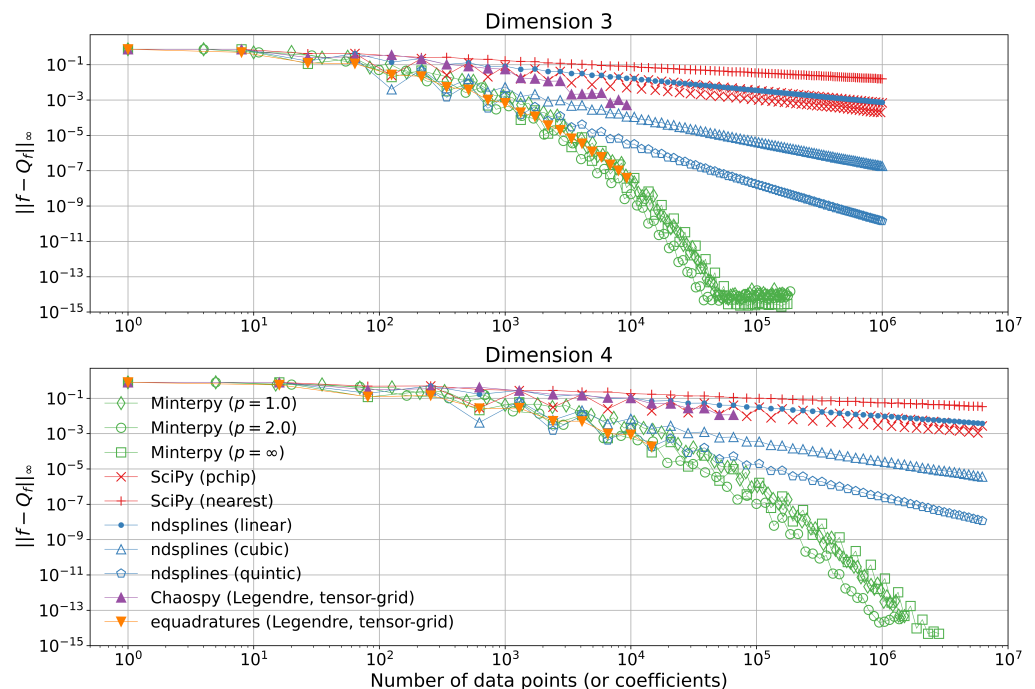

Figure 1: The comparison of Minterpy interpolating polynomials, approximating the Runge function in dimension $m = 3, 4$, with alternative methods from designated packages.

Figure 1 shows the accuracy of Minterpy interpolating polynomials for three different ℓ_p-degrees in dimension $m = 3, 4$[7]. The horizontal axis shows the number of coefficients (and

[7]Details of the numerical experiments can be found in (Wicaksono et al., 2025).

function evaluations), directly linked to the polynomial degree, to enable comparisons with other methods. The infinity norm of the difference between the function and its approximation,

$$\|f - Q_f\|_\infty = \max_{\boldsymbol{x} \in [-1,1]^m} |f(\boldsymbol{x}) - Q_f(\boldsymbol{x})|$$

is measured at $1'000'000$ random points.

The figure compares data-driven methods (SciPy v1.13.1, ndsplines v0.2.0post0) and pseudo-spectral methods (Chaospy v4.3.18, Equadratures v10). In the data-driven methods, approximation complexity is fixed as data increases. While ndsplines supports higher degrees, splines above degree 5 are rare in practice. The pseudo-spectral methods approximate functions using Legendre polynomial expansions on tensor-product grids, with coefficients computed via numerical integration. The coefficient count matches Minterpy interpolating polynomials with $p = \infty$. Equadratures, whose results are comparable to Minterpy, (softly) limits multi-index cardinality to 5×10^4 due to computational expense, while Chaospy struggles with tensor-product grids[8].

The results show that Minterpy polynomials provide highly accurate function approximation, demonstrating numerical stability and convergence down to 10^{-14}, and outperforming selected competing tools. However, global polynomials are generally more computationally expensive to evaluate than local piecewise polynomials or B-splines, as they require more floating-point operations. There are two primary reasons for this. First, global polynomials lack compact support; evaluating them typically involves computing all terms (i.e., coefficient-basis function pairs) in the expansion. Second, they often require high polynomial degrees, resulting in a large number of terms compared to local methods, which usually employ low-degree polynomials. Moreover, the basis functions in a high-degree global polynomial involve numerous multiplications, further increasing the computational cost.

Operations on the Minterpy polynomials

As mentioned, Minterpy polynomials support arithmetic operations (addition, subtraction, multiplication) and calculus operations (differentiation, definite integration). Except for definite integration (yielding a numerical value), these operations produce another polynomial, ensuring closure. Among compared tools, only Chaospy offers similar capabilities.

Polynomial regression

By default, Minterpy uses Leja-ordered Chebyshev-Lobatto nodes. For scattered or equispaced data, it supports well-conditioned least-squares construction (Veettil et al., 2022). The resulting polynomials are Minterpy polynomials[9].

Applications

Minterpy has been applied in various research fields, including data fitting in physics (Dornheim et al., 2023), serving as a surrogate model in blackbox optimization (Schreiber et al., 2023), and representing level sets in differential geometry (Veettil et al., 2023).

Author contributions

The contributions to this paper are listed according to CRediT. **D. Wicaksono**: Conceptualization, software, validation, visualization, writing—original draft. **U. Hernandez Acosta**: Conceptualization, project administration, software, writing—review & editing. **S. K. Thekke Veettil**: Conceptualization, software. **J. Kissinger**: Conceptualization, software. **M. Hecht**: Conceptualization, supervision, funding acquisition, writing—review & editing.

---

[8]Both Chaospy and equadratures support sparse polynomial construction, which can help reduce the number of coefficients. Comparing these approaches, however, is beyond the scope of this work.

[9]The polynomials are, however, not strictly interpolatory, i.e., they generally do not satisfy Equation 1.

Acknowledgments

References

Breuß, M., Kemm, F., & Vogel, O. (2018). A numerical study of Newton interpolation with extremely high degrees. *Kybernetika*, 279–288. https://doi.org/10.14736/kyb-2018-2-0279

Cools, R. (2002). Advances in multidimensional integration. *Journal of Computational and Applied Mathematics*, *149*(1), 1–12. https://doi.org/10.1016/s0377-0427(02)00517-4

de Branges, L. (1959). The Stone-Weierstrass theorem. *Proceedings of the American Mathematical Society*, *10*(5), 822–824. https://doi.org/10.1090/s0002-9939-1959-0113131-7

Dornheim, T., Wicaksono, D., Suarez-Cardona, J. E., Tolias, P., Böhme, M. P., Moldabekov, Z. A., Hecht, M., & Vorberger, J. (2023). Extraction of the frequency moments of spectral densities from imaginary-time correlation function data. *Physical Review B*, *107*(15), 155148. https://doi.org/10.1103/physrevb.107.155148

Driscoll, T. A., Hale, N., & Trefethen, L. N. (Eds.). (2014). *Chebfun guide*. Pafnuty Publications. https://www.chebfun.org/docs/guide/

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

Goldstine, H. H. (1977). *A history of numerical analysis from the 16th through the 19th century*. Springer New York. https://doi.org/10.1007/978-1-4684-9472-3

Hecht, M., Hofmann, P.-A., Wicaksono, D., Hernandez Acosta, U., Gonciarz, K., Kissinger, J., Sivkin, V., & Sbalzarini, I. F. (2025). *Multivariate Newton interpolation in downward closed spaces reaches the optimal geometric approximation rates for Bos–Levenberg–Trefethen functions*. https://arxiv.org/abs/2504.17899

Hegemann, N., & Heidenreich, S. (2023). PyThia: A Python package for uncertainty quantification based on non-intrusive polynomial chaos expansions. *Journal of Open Source Software*, *8*(89), 5489. https://doi.org/10.21105/joss.05489

Margolis, B., & Lyons, K. (2019). ndsplines: A Python library for tensor-product B-Splines of arbitrary dimension. *Journal of Open Source Software*, *4*(42), 1745. https://doi.org/10.21105/joss.01745

Meijering, E. (2002). A chronology of interpolation: From ancient astronomy to modern signal and image processing. *Proceedings of the IEEE*, *90*(3), 319–342. https://doi.org/10.1109/5.993400

Olver, S., Townsend, A., & others. (2023). ApproxFun.jl: A Julia package for function approximation. In *GitHub repository*. GitHub. https://github.com/JuliaApproximation/ApproxFun.jl

Reichel, L. (1990). Newton interpolation at Leja points. *BIT*, *30*(2), 332–346. https://doi.org/10.1007/bf02017352

Richardson, M., & others. (2024). ChebPy - a Python implementation of Chebfun. In *GitHub*

repository. GitHub. https://github.com/chebpy/chebpy

Schreiber, J., Wicaksono, D., & Hecht, M. (2023). Minimizing black boxes due to polynomial-model-based optimization. *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, *2*, 759–762. https://doi.org/10.1145/3583133.3590743

Seshadri, P., & Parks, G. (2017). Effective-Quadratures (EQ): Polynomials for computational engineering studies. *The Journal of Open Source Software*, *2*(11), 166. https://doi.org/10.21105/joss.00166

Swierczewski, C., & Verdier, O. (2024). pychebfun - Python Chebyshev functions. In *GitHub repository*. GitHub. https://github.com/pychebfun/pychebfun

Tal-Ezer, H. (1991). High degree polynomial interpolation in Newton form. *SIAM Journal on Scientific and Statistical Computing*, *12*(3), 648–667. https://doi.org/10.1137/0912034

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

Trefethen, L. N. (2011). Six myths of polynomial interpolation and quadrature. *Mathematics Today*, *47*, 184–188.

Trefethen, L. N. (2016). Inverse Yogiisms. *Notices of the American Mathematical Society*, *63*(11), 1281–1285. https://doi.org/10.1090/noti1446

Trefethen, L. N. (2017a). Cubature, approximation, and isotropy in the hypercube. *SIAM Review*, *59*(3), 469–491. https://doi.org/10.1137/16m1066312

Trefethen, L. N. (2017b). Multivariate polynomial approximation in the hypercube. *Proceedings of the American Mathematical Society*, *145*(11), 4837–4844. https://doi.org/10.1090/proc/13623

Trefethen, L. N. (2019). *Approximation theory and approximation practice, extended edition*. Society for Industrial; Applied Mathematics (SIAM).

Veettil, S. K. T., Zavalani, G., Acosta, U. H., Sbalzarini, I. F., & Hecht, M. (2023). Global polynomial level sets for numerical differential geometry of smooth closed surfaces. *SIAM Journal on Scientific Computing*, *45*(4), A1995–A2018. https://doi.org/10.1137/22m1536510

Veettil, S. K. T., Zheng, Y., Hernandez Acosta, U., Wicaksono, D., & Hecht, M. (2022). *Multivariate polynomial regression of Euclidean degree extends the stability for fast approximations of Trefethen functions*. arXiv. https://doi.org/10.48550/arXiv.2212.11706

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Weise, K., Poßner, L., Müller, E., Gast, R., & Knösche, T. R. (2020). Pygpc: A sensitivity and uncertainty analysis toolbox for Python. *SoftwareX*, *11*, 100450. https://doi.org/10.1016/j.softx.2020.100450

Wicaksono, D. C., Hernandez Acosta, U., Thekke Veettil, S. K., Kissinger, J., & Hecht, M. (2025). *Data to "Minterpy: Multivariate polynomial interpolation in Python"*. https://doi.org/10.14278/rodare.3379

Xiu, D. (2009). Fast numerical methods for stochastic computations: A review. *Communications in Computational Physics*, *5*(2–4), 242–272.

Xiu, D., & Karniadakis, G. E. (2002). The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, *24*(2), 619–644. https://doi.org/10.1137/S1064827501387826