The Journal of Open Source Software

# PyMetric: A Geometry Informed Array Mathematics Package

**Eliza C. Diggins** [1¶] **and Daniel R. Wik** [1]

1 University of Utah Department of Physics and Astronomy, Salt Lake City, Utah, USA ¶ Corresponding author

**DOI:** 10.21105/joss.08901

**Software**

- Review
- Repository
- Archive

**Editor:** Vincent Knight

**Reviewers:**

- @jcorbino
- @anthbapt

**Submitted:** 20 June 2025
**Published:** 08 January 2026



## Summary

PyMetric is a lightweight Python library designed to streamline differential geometry and vector calculus operations in user-defined coordinate systems, with a focus on applications in astrophysics and computational physics. The library was originally created to provide a geometric backend for the Pisces project, an in-development, general purpose astrophysical modeling and initial conditions library, but has since grown into an independent library due to its size and complexity. In many physical modeling tasks, it is both natural and advantageous to work in non-Cartesian coordinate systems that align with the inherent symmetries of the system. These coordinate systems can feature complex geometric structure which makes the explicit handling of differential operations cumbersome. This is particularly true for exotic coordinate systems (e.g. homoeoidal coordinate systems). PyMetric provides a unified abstraction that decouples the underlying coordinate representation from the operations themselves, allowing users to accurately compute gradients, divergences, Laplacians, and related geometric quantities through a consistent (and coordinate system agnostic) interface. This makes it easier to prototype and scale models in complex geometries without having to rewrite operations for each coordinate system.

The core design of PyMetric relies on a hybrid symbolic-numeric model that balances efficiency, flexibility, and accuracy. Symbolic computation is used to derive key geometric quantities, such as metric tensors, Christoffel symbols, and Jacobians, from a minimal set of coordinate system properties. Once generated, these symbolic structures can be converted into efficient numerical routines that operate on array-backed data and are composed to perform higher-level operations. This approach allows PyMetric to support coordinate-aware computation with minimal overhead, avoiding the need for repeated symbolic manipulation during runtime, while maintaining high accuracy through analytically correct geometric expressions. The result is a powerful and extensible framework that enables NumPy-style (Harris et al., 2020) workflows in complex coordinate systems without sacrificing physical fidelity.

In addition to its symbolic-numeric foundation, PyMetric provides structured abstractions for grids and field data, supporting a range of coordinate systems and buffer backends. Because the PyMetric field abstraction is only minimally coupled to the underlying data storage, it can interface with a variety of array backends, including in-memory arrays and HDF5 (The HDF Group, n.d.) storage for lazy-loading and chunked computation. This design enables coordinate-aware operations to be applied efficiently to large, multidimensional datasets without compromising generality or performance.

By automating core geometric operations across coordinate systems, PyMetric simplifies the development of physics-based modeling software that requires flexible geometric handling. Its design supports a broad spectrum of scientific computing applications, from simulating relativistic fluids to analyzing gravitational fields. In doing so, PyMetric establishes a modern and extensible foundation for geometry-aware computation in Python, enabling the creation of

accurate, efficient, and scalable models in complex coordinate geometries.

## Statement of need

Modern astrophysical modeling requires a high degree of flexibility—both in physical assumptions and in computational infrastructure. The Pisces Project (of which PyMetric is a part) is a general-purpose model-building framework for astrophysics that aims to unify and extend existing tools for generating models and initial conditions (e.g., DICE (Perret, 2016), GALIC (Yurin & Springel, 2014)) under a common, modular API. Its goal is to make it easier to construct complex, physically motivated models of systems such as galaxies, black holes, or relativistic fluids by exposing a simple and extensible interface for defining models, fields, and dynamics.

A persistent challenge in building such extensible modeling tools is the limited and inconsistent support for coordinate systems found in most existing software. Codes like EinsteinPy (EinsteinPy Development Team, 2024), DICE, and yt (Turk et al., 2010) often hard-code assumptions about coordinate geometry, making them difficult to generalize to new physical contexts or non-Cartesian coordinate systems. This lack of abstraction limits reusability and complicates the construction of unified modeling workflows across domains such as general relativity, galactic dynamics, and fluid mechanics.

To address this limitation, PyMetric was developed to be a lightweight library that standardizes coordinate-aware geometric computation. The library is designed to serve as the geometric backend for Pisces and similar modeling systems. It provides a consistent abstraction layer for defining coordinate systems, computing differential geometric quantities, and evaluating operators like gradients, divergences, and Laplacians; all without requiring the user to manage low-level details of tensor algebra or coordinate transformations.

PyMetric emphasizes extensibility and modularity through four core interfaces:

- **Coordinate System API** – Enables the definition and use of arbitrary coordinate systems with minimal required knowledge, while supporting symbolic derivation of metric-dependent quantities.
- **Buffer API** – Provides a backend-agnostic interface for array storage, allowing seamless integration with systems like HDF5, XArray, Dask, and unit-aware arrays.
- **Differential Geometry API** – Implements low-level, coordinate-independent formulations of core operations such as gradients, divergences, Laplacians, and volume elements.
- **Grid and Field API** – Supports flexible discretization strategies and a variety of field types, including sparse and dense scalar, vector, and tensor fields.

Together, these abstractions form a unified symbolic-numeric pipeline that allows high-level modeling code to operate naturally across diverse geometries and data representations. By standardizing geometric computation and decoupling it from specific coordinate assumptions or backend implementations, PyMetric addresses a longstanding gap in scientific Python infrastructure. This foundation enables Pisces to offer a powerful, composable, and user-friendly environment for building physically accurate models in astrophysics and beyond.

## Methodology

The core methodology behind PyMetric centers on a mathematically rigorous yet computationally practical framework for performing differential geometry operations in arbitrary coordinate systems. The library is designed to support seamless transitions between symbolic derivation and numerical evaluation, allowing for precise, efficient, and geometry-aware modeling.

A coordinate system in PyMetric is defined minimally by:

- A set of axes labels $(x^1, x^2, ...)$,

- Forward and inverse transformations between these coordinates and Cartesian Space $T(x, y, z)$ and $T^{-1}(x^1, x^2, x^3)$.
- A symbolically defined metric tensor $g_{\mu\nu}$.

From this core specification, PyMetric constructs key geometric quantities, such as the inverse metric $g^{\mu\nu}$, the metric density $\sqrt{g}$, and terms appearing in differential operations, such as $L^\nu = g^{-1/2}\partial_\mu(g^{1/2}g^{\mu\nu})$, which appears in the scalar Laplacian $\nabla^2\phi = L^\nu\partial_\nu\phi + g^{\mu\nu}\partial^2_{\mu\nu}\phi$. These are represented both as symbolic expressions-using SymPy (Meurer et al., 2017)-and as NumPy-backed callables. These quantities are computed lazily: they are only derived when required for a specific operation, avoiding unnecessary overhead.

Coordinate systems are categorized into types (e.g., orthogonal or curvilinear) that determine how symbolic properties are derived and which simplifications may apply. This abstraction enables users to model highly symmetric systems (e.g., spherical or ellipsoidal coordinates) as easily as more general curvilinear systems.

## Field and Grid Operations

Fields in PyMetric are array-backed data structures (typically NumPy or HDF5 buffers) that are explicitly associated with a coordinate system and grid. The grid handling supports flexible discretization strategies, including cell-centered and node-centered layouts, as well as ghost zones for finite-difference operations. Support is currently provided for single-grid configurations with arbitrary spacing; however, multigrid extension is planned for future releases. While fields behave like standard NumPy arrays, they also carry metadata about their geometric context, including coordinate labels, spacing, and metric-aware tensor properties.

Operations on fields, such as computing covariant derivatives, applying Laplacians, or transforming between bases, are automatically dispatched to appropriate symbolic expressions and numerical kernels based on the field's variance and the geometry of the underlying coordinate system.

This design allows users to write high-level, reusable code that is agnostic to the specific geometry, while still benefiting from the mathematical correctness and efficiency of coordinate-aware computation.

## Future Development

The development roadmap for PyMetric is focused on deepening its mathematical capabilities and expanding its utility in advanced physical modeling contexts, particularly those involving curved and relativistic spacetimes. While the current implementation supports a robust suite of differential operators in orthogonal and curvilinear coordinate systems, several avenues for future growth are planned:

1. Expanded Differential Operator Support:

   PyMetric will be extended to support a broader range of tensor calculus operations, including:

   - Covariant derivatives of higher-rank tensors, enabling modeling of tensor transport and geodesic deviation.
   - Tensor contractions and curvature operations, including the Riemann, Ricci, and Einstein tensors, to support simulations in general relativity and cosmology.

These features will allow PyMetric to serve as a general-purpose differential geometry engine suitable for high-fidelity modeling in physics, engineering, and applied mathematics.

2. Relativistic and Non-Flat Coordinate Systems

- A key area of expansion is support for relativistic geometries, where the metric tensor is no longer positive-definite and may depend dynamically on spacetime coordinates. Planned features include:
- General Lorentzian manifolds, including Schwarzschild, Kerr, and FLRW spacetimes, enabling direct modeling of astrophysical systems governed by Einstein's field equations.

PyMetric is explicitly intended as a modeling and analysis tool, not a time-domain simulation engine. It provides geometric infrastructure for constructing and analyzing equations defined on curved spacetimes, but does not aim to solve dynamical systems or perform numerical integration of time-evolving fields.

## Usage Example

To demonstrate the basic capabilities of the Pymetric library, we include a simple example of the typical workflow computing the Laplacian ($\nabla^2$) of a field in spherical coordinates. We use $F(r, \theta) = r\cos(\theta)$ as our test function, which has a known Laplacian of zero. A visualization of $F(r, \theta)$ and its Laplacian is shown in Figure 1, demonstrating the library's ability to perform geometry-aware computations directly on array data.

```
import pymetric as pym
import numpy as np

# Define spherical coordinate system and grid
cs = pym.coordinates.SphericalCoordinateSystem()
grid = pym.grids.GenericGrid(
    cs,
    [
        np.linspace(0.1, 4.9, 300),                # r
        np.linspace(0.01, np.pi - 0.01, 100),      # theta
        np.linspace(0.01, 2 * np.pi - 0.01, 100),  # phi
    ],
    center="cell",
    bbox=[(0, 5), (0, np.pi), (0, 2 * np.pi)],
    ghost_zones=2,
)

# Define scalar field F(r, theta) = r * cos(theta)
# This is a good test case since Lap(F) = 0.
field = pym.DenseField.from_function(
    lambda r, theta: r * np.cos(theta),
    grid,
    axes=["r", "theta"],
)

# Compute Laplacian
F_lap = field.element_wise_laplacian()
```

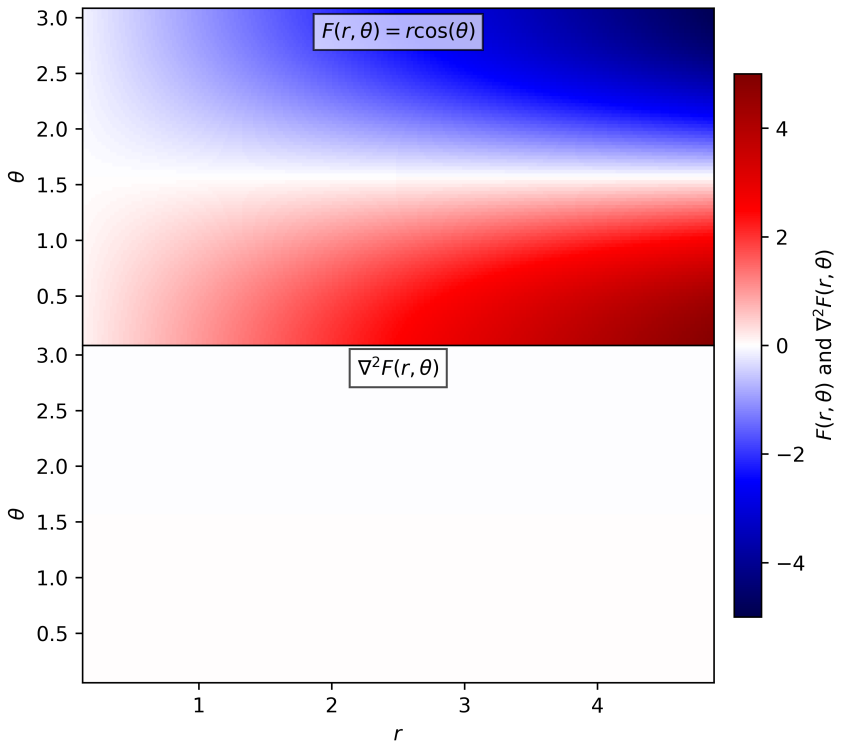


**Figure 1:** A scalar field $F(r, \theta) = r\cos(\theta)$ and its Laplacian $\nabla^2 F(r, \theta)$, computed in spherical coordinates using PyMetric.

## References

EinsteinPy Development Team. (2024). *EinsteinPy: Python library for general relativity*. https://doi.org/10.48550/arXiv.2005.11288

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., … Scopatz, A. (2017). SymPy: Symbolic computing in python. *PeerJ Computer Science*, *3*, e103. https://doi.org/10.7717/peerj-cs.103

Perret, V. (2016). DICE: Disk initial conditions environment. *Astrophysics Source Code Library*, ascl–1607.

The HDF Group. (n.d.). *Hierarchical Data Format, version 5*. https://github.com/HDFGroup/hdf5

Turk, M. J., Smith, B. D., Oishi, J. S., Skory, S., Skillman, S. W., Abel, T., & Norman, M. L. (2010). Yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, *192*(1), 9. https://doi.org/10.1088/0067-0049/192/1/9

Yurin, D., & Springel, V. (2014). GALIC: Galaxy initial conditions construction. *Astrophysics Source Code Library*, ascl–1408. https://doi.org/10.48550/arXiv.1402.1623