

aurel: A Python package for automatic relativistic calculations

Robyn L. Munoz¹, Christian T. Byrnes¹, and Will J. Roper¹

¹ Department of Physics and Astronomy, University of Sussex, Brighton, BN1 9QH, United Kingdom

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

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

Editor: [Open Journals](#)

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

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

Summary

aurel is an open-source Python package designed to automatically calculate relativistic quantities. It uses an efficient, flexible and user-friendly caching and dependency-tracking system, ideal for managing the highly nonlinear nature of general relativity. The package supports both symbolic and numerical calculations. The symbolic part extends SymPy with additional tensorial calculations. The numerical part computes a wide range of tensorial quantities, such as curvature, matter kinematics and much more, directly from any spacetime and matter data arrays using finite-difference methods. Inputs can be either generated from analytical expressions or imported from Numerical Relativity (NR) simulations, with helper functions provided to read in data from standard NR codes. Given the increasing use of NR, aurel offers a timely post-processing tool to support the popularisation of this field.

Statement of need

General relativity describes matter as moving according to how distances shrink or expand; likewise, the intervals of space and time evolve depending on the distribution of matter. Handling this dynamic “mesh” of distances and times requires elaborate tensor algebra that, in some cases, can only be managed with symbolic or numerical tools. Naturally, NR has become essential for modern astrophysics, cosmology, and gravitational physics, most notably in the modelling of gravitational-wave signals.

While established computational frameworks focus on solving and evolving Einstein’s field equations, with specific key diagnostics, they leave calculations of the remaining analysis to the discretion of the researchers. Newcomers to the field then face a substantial overhead until they develop their own personal post-processing codes. Established researchers also face the tedious task of handling intermediary variables and indices when calculating new quantities. The field then suffers from this error-prone, time-consuming process and would benefit from an accessible, open-source, standardised framework to automate these steps.

We therefore present aurel, an open-source Python package designed to streamline relativistic calculations. It is hosted on [GitHub](#) and is available on [PyPI](#). The documentation is available through [GitHub Pages](#).

State of the field

When looking for general relativity Python packages, there are a number of tools that provide symbolic calculations ([Bapat et al., 2020](#); [Czaja, n.d.](#); [Della Monica, 2025](#); [Gourgoulhon et al., 2015](#); [Hackstein & Hackmann, 2025](#); [Martín & Sureda, 2022](#); [Shoshany, 2025](#); [Wittig & Grover, 2017](#)). Or, one may also consider computer algebra systems ([Maplesoft, 2025](#); [Martín-García](#)

38 & others, 2025; Wolfram Research, 2025). However, when non-linearities become too complex
39 for symbolic packages, NR is used instead.

40 Einstein Toolkit (Löffler & others, 2012; Rizzo et al., 2025) is a large community-driven
41 software whose tools enable the evolution of Einstein's field equations. Diagnostic and further
42 analysis calculations are typically performed on the fly, during simulations. To study the
43 outputs, provided by Carpet, there are Python reading packages available (Bozzola, 2021;
44 Ferguson et al., 2025; Kastaun, n.d.; Radice, n.d.). These extra calculations can slow down
45 the simulation of the spacetime evolution, and if certain relativistic quantities are not available
46 in Einstein Toolkit, or in one of the post-processing packages, then the user needs to code
47 that up themselves.

48 There are a number of other well-established NR codes (Andrade et al., 2021; Barrera-Hinojosa
49 & Li, 2020; Palenzuela et al., 2025; Wright, 2018; Zhang et al., 2025) that also have their
50 own diagnostic tools. However, these are typically built-in, so going from one code to another,
51 to benchmark or to use their different types of applications, requires learning the ecosystem of
52 each.

53 To improve the community's versatility and limit the repeated implementation of error-prone
54 calculations, there is a motivation to provide packages for computing relativistic quantities in
55 an NR-code-agnostic way. Especially in the post-processing sense, where all calculations are
56 done from a given NR spacetime and matter solution. A couple of notable packages (Grasso
57 et al., 2021; Pook-Kolb et al., 2019) focus on ray tracing, or apparent-horizon finding, which
58 are currently beyond the scope of aurel. While others have more overlap (Cranganore et al.,
59 2025; Munoz & Bruni, 2023) in calculating curvature terms, they differ in scope and workflow.

60 Here, aurel innovates in its automatic design, which is easily extendable and provides flexibility
61 and robustness with a large and ever-growing catalogue of relativistic quantities. A precursor
62 to this package was EBWeyl (Munoz & Bruni, 2023), as it provided calculations of gravito-
63 electromagnetic contributions from base spacetime and matter quantities. aurel now has
64 a completely different structure (relying on the automatic dependency resolution), provides
65 calculations of many more terms, over time, and has entirely new features as described in the
66 following section.

67 Software Design

68 aurel provides an intuitive interface for the automatic calculation of general relativistic
69 quantities, either symbolically (with AurelCoreSymbolic, built on SymPy (Meurer et al., 2017))
70 or numerically (with AurelCore, which heavily utilises numpy.einsum (Harris et al., 2020) for
71 efficient operations on array data structures).

72 Both require base quantities such as the spacetime coordinates or the parameters of the
73 Cartesian numerical grid, as well as the spacetime and matter distributions (the Minkowski
74 vacuum is otherwise assumed). These inputs can either come from analytical expressions, with
75 a couple of built-in solutions available, or from output data from any NR simulations; they
76 just need to be passed as numpy arrays.

77 Specifically, for simulations run with Carpet in the Einstein Toolkit, the reading module
78 provides helper functions to load and organise the 3D data. These can read the parameter file,
79 summarise available iterations and variables, and handle data separated across restarts, chunks,
80 or refinement levels for normal Carpet data files or checkpoint files. To speed up repeated
81 data reading, read_data can also split the data per iteration, instead of per variable.

82 Then, once input data is provided, users can directly request a wide range of relativistic
83 quantities, including: spacetime; matter (Eulerian, Lagrangian, or conserved); NR formulations;
84 constraints; fluid covariant kinematics; null ray expansion; 3- and 4-dimensional curvature;
85 gravito-electromagnetism; Weyl scalars and invariants (including gravitational waves). To see
86 a full list of available quantities, see: [descriptions](#). Tools are also provided for spatial and

87 spacetime covariant derivatives and Lie derivatives. All spatial derivatives are computed by the
88 FiniteDifference class that provides 2nd, 4th, 6th and 8th order schemes, using periodic,
89 symmetric or one-sided boundary conditions.

90 Automatic Computational Pathway

91 The aurel automatic process composes a computational pathway at runtime to evaluate
92 the requested quantities. This is implemented through a lazy-evaluation memoised property
93 pattern, where each quantity is defined as a method of the core class that may depend on
94 other quantities. This design has been chosen for its flexibility and accessibility while remaining
95 robust under future extensions.

96 Quantities are requested via a user-friendly dictionary-style access, e.g. `rel["s_RicciS"]`,
97 which triggers the lazy memoised check to see if this is already cached. If yes, then the result is
98 directly returned. If not, then the corresponding method is called, which recursively triggers the
99 calculation of dependencies (e.g. `rel["s_Ricci_down3"]`). This continues until the requested
100 quantities can be calculated and so returned.

101 To avoid redundant computations, each result is cached, which builds up a cache memory that
102 needs to be efficiently managed. So, inspired by Python's garbage collection, aurel uses an
103 intelligent eviction policy that tracks memory footprint, evaluation counts, and last-access
104 times. When the configurable thresholds are exceeded, the older and heavier cached quantities
105 are removed, while safeguarding protected base quantities. Throughout this process, aurel
106 keeps the user informed on progress by providing verbose updates on the computation and
107 caching workflow.

108 Time dependence

109 All calculations within the AurelCore class are evaluated at a single fixed time, corresponding
110 to one slice in time, so that individual time steps can be treated independently. For multiple
111 time steps, an AurelCore object needs to be created and the requested quantities collected for
112 each.

113 To streamline this process, aurel provides the `over_time` function to do exactly this, and
114 also compute summary statistics over the grid domain (e.g., max/min) at each time step. By
115 design, it is easily extensible, so `over_time` also accepts custom functions of new relativistic
116 quantities and summary statistics. This makes aurel versatile, supporting an infinite number
117 of ways to view a problem and develop diagnostic tools.

118 Research Impact Statement

119 aurel is a specialist tool for general relativity researchers and streamlines numerical relativists'
120 post-processing workflow. Through conference interactions and collaborations involving the
121 authors, this package has gradually been disseminated to individual researchers who appreciate
122 the effortless integration, satisfying dependency resolution and substantial reduction to post-
123 processing overhead. Indeed, in ongoing studies involving NR simulations of primordial
124 black hole formation, aurel has increased capacity and redirected repetitive and error-prone
125 development efforts towards exploring a broader range of simulated scenarios. Additionally, for
126 master students, the straightforward and transparent design has provided an easy gateway for
127 them to analyse NR simulations and so quickly get results within the duration of their projects.
128 Going forward, awareness of this code will build upon publication, reaching a wider audience
129 and supporting the popularisation of NR.

130 AI usage disclosure

131 GitHub Copilot Claude Sonnet 4 was used for the development and documentation of this
 132 package. Autocompletion suggestions were accepted via the VSCode Copilot plugin, and upon
 133 the developer's request, edits and code snippets were generated via the large language model's
 134 user interface. The most significant AI contributions came in drafting the docstrings and
 135 scaffolding the test suite, both of which are essential for the accessibility and robustness of this
 136 package. Each and every suggestion or contribution was meticulously reviewed and adjusted
 137 before being included by the authors, who made all core design decisions and innovated the
 138 original structural concept. Finally, this paper was prepared without the use of generative
 139 language models, solely with grammar checkers.

140 Acknowledgements

141 We thank Nat Kemp for being one of the first testers of aurel. We thank Ian Hawke for
 142 support and suggestions.

143 RM and WR are supported by an STFC grant ST/X001040/1. CB is supported by STFC
 144 grants ST/X001040/1 and ST/X000796/1.

145 Andrade, T., Salo, L. A., Aurrekoetxea, J. C., Bamber, J., Clough, K., Croft, R., Jong, E.
 146 de, Drew, A., Duran, A., Ferreira, P. G., Figueras, P., Finkel, H., França, T., Ge, B.-X.,
 147 Gu, C., Helfer, T., Jäykkä, J., Joana, C., Kunesch, M., ... Wong, K. (2021). GRChombo:
 148 An adaptable numerical relativity code for fundamental physics. *Journal of Open Source*
 149 *Software*, 6(68), 3703. <https://doi.org/10.21105/joss.03703>

150 Bapat, S., Saha, R., Bhatt, B., Jain, S., Jain, A., Vela, S. O., Khandelwal, P., Shivottam, J.,
 151 Ma, J., Ng, G. S., Kerhalkar, P., Sarode, H. S., Sharma, R., Gupta, M., Gupta, D., Tyagi,
 152 T., Rustagi, T., Singh, V., Bansal, S., ... Chan, M. Y. (2020). *EinsteinPy: A community*
 153 *python package for general relativity*. <https://arxiv.org/abs/2005.11288>

154 Barrera-Hinojosa, C., & Li, B. (2020). GRAMES: A new route to general relativistic *N*-body
 155 simulations in cosmology. Part i. Methodology and code description. *JCAP*, 01, 007.
 156 <https://doi.org/10.1088/1475-7516/2020/01/007>

157 Bozzola, G. (2021). Kuibit: Analyzing einstein toolkit simulations with python. *Journal of*
 158 *Open Source Software*, 6(60), 3099. <https://doi.org/10.21105/joss.03099>

159 Cranganore, S. S., Bodnar, A., Berzins, A., & Brandstetter, J. (2025). *Einstein fields: A neural*
 160 *perspective to computational general relativity*. <https://arxiv.org/abs/2507.11589>

161 Czaja, W. (n.d.). *GraviPy, tensor calculus package for general relativity*. <https://pypi.python.org/pypi/GraviPy>

163 Della Monica, R. (2025). PyGRO: A python integrator for general relativistic orbits. *Astronomy*
 164 *& Astrophysics*, 698, A193. <https://doi.org/10.1051/0004-6361/202554300>

165 Ferguson, D., Anne, S., Gracia-Linares, M., Iglesias, H., Jan, A., Martinez, E., Lu, L., Meoni, F.,
 166 Nowicki, R., Trostel, M., Tsao, B.-J., & Valorz, F. (2025). *Mayawaves* (Version v2025.12).
 167 Zenodo. <https://doi.org/10.5281/zenodo.17981058>

168 Gourgoulhon, E., Beijer, M., & Mancini, M. (2015). Tensor calculus with open-source
 169 software: The SageManifolds project. *Journal of Physics: Conference Series*, 600(1),
 170 012002. <https://doi.org/10.1088/1742-6596/600/1/012002>

171 Grasso, M., Villa, E., Korzyński, M., & Matarrese, S. (2021). Isolating nonlinearities of
 172 light propagation in inhomogeneous cosmologies. *Phys. Rev. D*, 104(4), 043508. <https://doi.org/10.1103/PhysRevD.104.043508>

174 Hackstein, J. P., & Hackmann, E. (2025). GREOPy: A python package for solving the

- emitter-observer problem in general relativity. *Journal of Open Source Software*, 10(112), 8765. <https://doi.org/10.21105/joss.08765>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Kastaun, W. (n.d.). *PostCactus*. GitHub. <https://github.com/wokast/PyCactus>
- Löffler, F., & others. (2012). The einstein toolkit: A community computational infrastructure for relativistic astrophysics. *Class. Quant. Grav.*, 29, 115001. <https://doi.org/10.1088/0264-9381/29/11/115001>
- Maplesoft, a division of W. M. Inc. (2025). *Maple, version 2025.2*. <https://www.maplesoft.com/products/maple/>
- Martín, M. S., & Sureda, J. (2022). Pytearcats: PYthon TEnsor AlgeBRa calCulATor a python package for general relativity and tensor calculus. *Astronomy and Computing*, 100572. <https://doi.org/10.1016/j.ascom.2022.100572>
- Martín-García, J. M., & others. (2025). *xAct: Efficient tensor computer algebra for the wolfram language*. <https://www.xact.es/>
- 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>
- Munoz, R. L., & Bruni, M. (2023). EBWeyl: A code to invariantly characterize numerical spacetimes. *Classical and Quantum Gravity*, 40(13), 135010. <https://doi.org/10.1088/1361-6382/acd6cf>
- Palenzuela, C., Bezares, M., Liebling, S., Schianchi, F., Abalos, J. F., Aguilera-Miret, R., Bona, C., Carretero, J. A., Massò, J., Smith, M. P., Amponsah, K., Kornet, K., Miñano, B., Pareek, S., & Radia, M. (2025). *MHDuet : A high-order general relativistic radiation MHD code for CPU and GPU architectures*. <https://arxiv.org/abs/2510.13965>
- Pook-Kolb, D., Birnholtz, O., Krishnan, B., & Schnetter, E. (2019). Existence and stability of marginally trapped surfaces in black-hole spacetimes. *Phys. Rev. D*, 99(6), 064005. <https://doi.org/10.1103/PhysRevD.99.064005>
- Radice, D. (n.d.). *Scidata*. Bitbucket. <https://bitbucket.org/dradice/scidata/src/master/>
- Rizzo, M., Haas, R., Brandt, S. R., Etienne, Z., Ferguson, D., Sanches, L. T., Tsao, B.-J., Werneck, L., Boyer, D., Bozzola, G., Cheng, C.-H., Cupp, S., Diener, P., Jacques, T. P., Ji, L., Macpherson, H., Markin, I., Schnetter, E., Tichy, W., ... Zink, B. (2025). *The einstein toolkit* (The "Martin D. Kruskal" release, ET_2025_05). Zenodo. <https://doi.org/10.5281/zenodo.15520463>
- Shoshany, B. (2025). OGREPy: An object-oriented general relativity package for python. *Journal of Open Research Software*, 13. <https://doi.org/10.5334/jors.558>
- Wittig, A. N., & Grover, J. (2017). *PyHole: General relativity ray tracing and analysis tool*. <https://eprints.soton.ac.uk/453123/>
- Wolfram Research, Inc. (2025). *Mathematica, version 14.3*. <https://www.wolfram.com/mathematica>
- Wright, A. (2018). *AlexJamesWright/METHOD: Initial public release* (Version 1.0). Zenodo. <https://doi.org/10.5281/zenodo.1404697>

- 222 Zhang, H., Li, B., Weinzierl, T., & Barrera-Hinojosa, C. (2025). ExaGRyPE: Numerical general
223 relativity solvers based upon the hyperbolic PDEs solver engine ExaHyPE. *Comput. Phys.*
224 *Commun.*, 307, 109435. <https://doi.org/10.1016/j.cpc.2024.109435>

DRAFT