The Journal of Open Source Software

Antupy: A Python package for energy engineering simulations

David Saldivia[1]

1 Solar Energy Research Center (SERC) Chile.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 29 January 2026
Published: unpublished

Summary

antupy, from Mapuche language *antü*, meaning "sun"(Wikipedia contributors, 2025), is a Python package designed as a toolkit for energy system simulations. The package provides a framework organised around three type of classes. The core data types (`Var`, `Array`, `Frame`) for handling physical quantities with automatic unit conversion; simulation classes (`Model`, `Plant`, `Analyser`, `TimeSeriesGenerator`) that enable modular and extensible simulation workflows; and a suite of utility modules for thermophysical properties (`props`), heat transfer correlations (`htc`), and solar calculations (`solar`). Built on top of established scientific libraries including NumPy (Harris et al., 2020), pandas (The pandas development team, 2024) (with a polars migration as a future project), and SciPy (Virtanen et al., 2020). This paper focuses on the core unit management system, some of the utility modules, and the abstract protocol architecture that enables researchers to develop custom energy system models with well-structured code while maintaining dimensional consistency throughout their simulations and post-processing.

Statement of need

This software targets (energy/mechanical) engineering research and education. From undergraduates taking their first basic science courses to active researchers, computational tools that balance accessibility, flexibility, and rigor are essential for solving engineering problems. Energy and mechanical engineering programs increasingly rely on computational methods to teach thermodynamic cycles, heat transfer, and renewable energy systems. Simultaneously, researchers working on solar energy deployment need flexible frameworks to prototype novel system configurations, conduct parametric studies, and validate experimental results. These two domains—education and research—share a common need for tools that are both pedagogically transparent and sufficiently powerful for real-world applications.

Established energy simulation platforms such as System Advisor Model (SAM) (National Renewable Energy Laboratory, 2024), Engineering Equation Solver (EES) (F-Chart Software, n.d.), and TRNSYS (University of Wisconsin-Madison, n.d.) have proven invaluable for industry applications and detailed system modeling. However, these tools present barriers for Python-based workflows, which have become the de facto standard in data science, machine learning, and modern scientific computing. While Python packages like TESPy (Witte et al., 2020) provide thermal system modeling capabilities and pvlib (Holmgren et al., 2018) excels at photovoltaic performance simulation, there remains a gap for a general-purpose framework specialized in annual energy simulations that enables researchers to implement custom modules, control solvers, and integrate diverse energy technologies within a unified architecture. Existing Python tools often focus on specific technologies or require significant overhead to extend beyond their original scope.

A fundamental challenge in energy system modeling is the management of physical units across calculations involving thermodynamics, heat transfer, and fluid mechanics. Engineers routinely

The Journal of Open Source Software

work with temperatures in Celsius and Kelvin, pressures in Pascals and bar, heat transfer rates in Watts and kW, and must ensure dimensional consistency when combining thermophysical properties from sources like CoolProp, convection correlations, and solar radiation models. While Python packages for unit management exist—such as Astropy (Astropy Collaboration, 2013), Pint (Grecco, 2024), and forallpeople (Mead, 2024)—these tools do not seamlessly integrate unit variables across scalar (`Var`), vector (`Array`), and tabular (`Frame`) data structures. Furthermore, `antupy` employs simple, standard unit labels following intuitive rules (detailed in the documentation) that reduce cognitive overhead for engineers. By combining thermophysical property evaluation, heat transfer coefficient calculations, and solar geometry routines with automatic unit tracking and conversion, `antupy` provides a cohesive framework where physical quantities carry their units throughout the simulation workflow, reducing errors and improving code readability while integrating essential utilities for energy engineering into a single, coherent package.

Software design

The `antupy` package architecture (Figure (?)) is organized around core data structures with two derived functional groups. The **core layer** provides three immutable data types: `Var` for scalar physical quantities, `Array` for homogeneous vector data (based on `numpy.ndarray` class), and `Frame` (extending `pandas.DataFrame`) for tabular data with per-column unit tracking. These types support arithmetic operations with automatic unit conversion and dimensional checking—for example, adding `Var(5.0, "kg")` and `Var(500, "g")` correctly yields `5.5 [kg]`. Building on these core classes, the **utilities modules** provide domain-specific functionality: `props` (thermophysical properties via CoolProp and some own implementations), `htc` (heat transfer correlations for natural and forced convection), `solar` (sun position and radiation calculations), and `loc` (geographical location management with Australian and Chilean databases). The **protocol/abstract classes** define interfaces for extensibility: `Model` (component-level solvers), `TimeSeriesGenerator` (weather and market data), `Plant` (system integration), and `Analyser` (parametric studies). Both the utilities and protocols are designed to be fully compatible with the core unit-aware data structures.

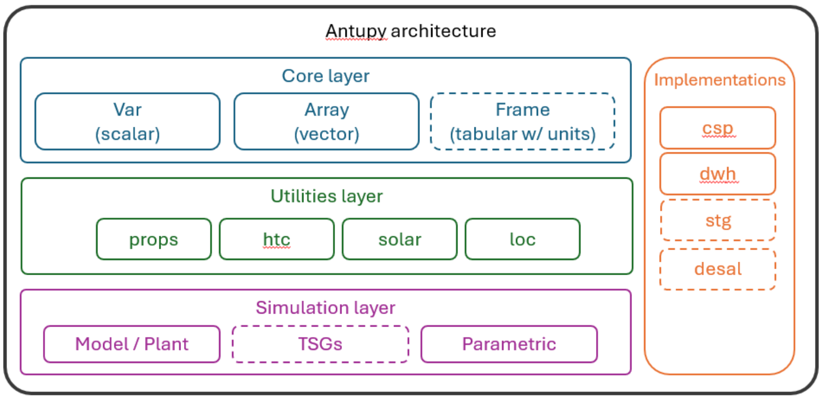

Figure 1: Architecture of the antupy package.

The package is designed for minimal friction in typical workflows. A simple example calculating heat stored in a domestic water heater from a cold tank until full hot tank, demonstrates the unit-aware approach:

```
from antupy import Var
from antupy.props import Water

temp_max = Var(60, "degC")
```

```python
temp_mains = Var(20, "degC")
vol_tank = Var(300, "L")
fluid = Water()

temp_avg = (temp_high+temp_mains)/2
cp = fluid.cp(temp_avg)
rho = fluid.rho(temp_avg)

q_stg = vol_tank * rho * cp * (temp_high - temp_mains)
q_stg = q_stg.su("kWh")

print(f"Energy stored: {q_stg:.1f}")
```

The `Parametric` analyser enables sensitivity studies by automatically managing parameter combinations and preserving units in results tables (stored as `Frame` objects). Current `TimeSeriesGenerator` implementations include market price data for Australia and Chile, as well as weather data through TMY (Typical Meteorological Year) and historical weather datasets. Comprehensive documentation is available online, including detailed introductions to the core variables, introductory examples, usage guides for the utility libraries, and the complete API reference. The package requires Python 3.12.

Research Impact Statement

The antupy codebase represents a formalization and generalization of simulation frameworks developed during concentrated solar thermal (CST) and domestic water heating (DWH) research projects reported in several publications (Saldivia et al., 2021, 2025; Saldivia & Taylor, 2023). The original basecodes of these simulations were updated using the current version of antupy and are publicly available online DavidSaldivia (2025). Through these initial applications the core architecture was developed and the identified common patterns informed the protocols design. Current development priorities include updating other project-specific codebases, including desalination and energy storage applications Valdivia et al. (2020). Finally, future enhancements will expand the `TimeSeriesGenerator` ecosystem to include additional weather data sources, electricity market price signals, and load profile generators for diverse human behaviour such as electricity or domestic hot water consumption. Community contributions are welcomed through the project's GitHub repository, with emphasis on maintaining the balance between educational clarity and research-grade capability that defines antupy's design philosophy.

AI usage disclosure

All the main classes were coded without any AI assistance, except by `Frame` and `Plant`, that were developed with assistance of VSCode-integrated Claude Sonnet 4.5. The same tool was used to develop part of the tests and the main classes docstrings using an iterative process.

Acknowledgments

The author expresses gratitude to the project ANID/FONDAP/1523A0006 "Solar Energy Research Center"—SERC-Chile. Additionally, with the name, the author acknowledges the Mapuche people and its worldview as an inspiration. The first beta version of this codebase was written in Temuco, in Mapuche's heartland.

References

Astropy Collaboration. (2013). Astropy: A community Python package for astronomy. In *Astronomy & Astrophysics* (Vol. 558, p. A33). https://doi.org/10.1051/0004-6361/201322068

DavidSaldivia. (2025). *DavidSaldivia/bdr_csp*. https://github.com/DavidSaldivia/bdr_csp

F-Chart Software. (n.d.). *Engineering Equation Solver (EES)*. https://fchartsoftware.com/ees/.

Grecco, H. E. (2024). *Pint: makes units easy*. https://pint.readthedocs.io/

Harris, C. R., Millman, K. J., Walt, S. J. van der, & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Holmgren, W. F., Hansen, C. W., & Mikofski, M. A. (2018). pvlib python: A python package for modeling solar energy systems. *Journal of Open Source Software*, *3*(29), 884. https://doi.org/10.21105/joss.00884

Mead, C. F. (2024). *forallpeople: A Python package for handling physical quantities*. https://github.com/connorferster/forallpeople

National Renewable Energy Laboratory. (2024). *System Advisor Model (SAM)*. https://sam.nrel.gov/.

Saldivia, D., Bilbao, J., & Taylor, R. A. (2021). Optical analysis and optimization of a beam-down receiver for advanced cycle concentrating solar thermal plants. *Applied Thermal Engineering*, *197*, 117405. https://doi.org/10.1016/j.applthermaleng.2021.117405

Saldivia, D., Klisser, R., Saberi, H., Bruce, A., & Yildiz, B. (2025). Thermal characterization of domestic electric water heating systems as distributed storage. *Applied Thermal Engineering*, 129560. https://doi.org/10.1016/j.applthermaleng.2025.129560

Saldivia, D., Rosales, C., Barraza, R., & Cornejo, L. (2019). Computational analysis for a multi-effect distillation (MED) plant driven by solar energy in Chile. *Renewable Energy*, *132*, 206–220. https://doi.org/10.1016/j.renene.2018.07.139

Saldivia, D., & Taylor, R. A. (2023). A Novel Dual Receiver–Storage Design for Concentrating Solar Thermal Plants Using Beam-Down Optics. *Energies*, *16*(10), 4157. https://doi.org/10.3390/en16104157

The pandas development team. (2024). *Pandas-dev/pandas: pandas*. Zenodo. https://doi.org/10.5281/zenodo.3509134

University of Wisconsin-Madison. (n.d.). *TRNSYS - Transient System Simulation Tool*. http://www.trnsys.com/.

UNSW-CEEM/tm_solarshift. (2024). Collaboration on Energy; Environmental Markets (CEEM). https://github.com/UNSW-CEEM/tm_solarshift

Valdivia, P., Barraza, R., Saldivia, D., Gacitúa, L., Barrueto, A., & Estay, D. (2020). Assessment of a Compressed Air Energy Storage System using gas pipelines as storage devices in Chile. *Renewable Energy*, *147*, 1251–1265. https://doi.org/10.1016/j.renene.2019.09.019

Virtanen, P., Gommers, R., Oliphant, T. E., & others. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wikipedia contributors. (2025). *Mapuche people*. https://en.wikipedia.org/wiki/Mapuche_people.

Witte, F., Hofmann, M., Meier, J., Tuschy, I., & Tsatsaronis, G. (2020). TESPy: Thermal

Engineering Systems in Python. *Journal of Open Source Software*, *5*(49), 2178. https://doi.org/10.21105/joss.02178