The Journal of Open Source Software

PyMatLib

Rahil Miten Doshi [1,2], **Matthias Markl** [2], **and Harald Koestler** [2,3]

1 Chair for System Simulation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany **2** Chair of Materials Science and Engineering for Metals, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany **3** Erlangen National High Performance Computing Center (NHR@FAU), Erlangen, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

PyMatLib is an extensible, open-source Python library that streamlines the definition and use of material properties in numerical simulations. The library allows users to define complex material behaviors-ranging from simple constants to experimental data -in human-readable YAML configuration files. These are automatically converted into symbolic mathematical expressions for direct use in scientific computing frameworks. PyMatLib supports both pure metals and alloys, offers six different property definition methods, and intelligently manages dependencies between different material properties. It is designed for high-performance computing applications, and serves as a seamless bridge between experimental data and numerical simulation, making sophisticated material modeling accessible to a broader scientific community.

Statement of Need

Accurate numerical simulation requires accounting for material properties—such as thermal conductivity, density, and heat capacity —that are not constant but depend on variables like temperature, pressure, or concentration (Lewis et al., 1996; Zienkiewicz et al., 2013). This challenge is compounded by the wide variation in data availability, from well-characterized models for established materials to sparse experimental points for novel alloys. Consequently, property definitions can range from simple constants to complex tabular datasets or sophisticated equations, creating a significant integration hurdle for researchers.

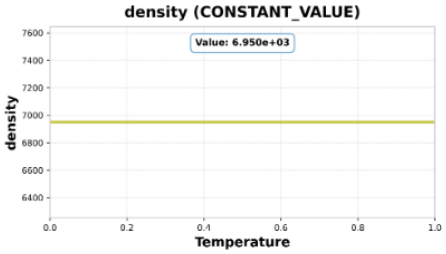
To manage this complexity, researchers often resort to manual interpolation, custom scripting, or proprietary software, which compromises reproducibility and standardization (Ashby, 2013). While valuable resources like the NIST WebBook (National Institute of Standards and Technology, 2023) and libraries such as CoolProp (Bell et al., 2014) exist, they typically provide raw data without the integrated processing needed to unify these varied formats. Similarly, specialized CALPHAD databases (Lukas et al., 2007) are powerful but often require proprietary software and do not easily integrate with general-purpose simulation codes.

This gap forces the development of ad-hoc solutions, hindering workflow efficiency and the adoption of FAIR data principles (Wilkinson et al., 2016). PyMatLib was created to bridge this gap by providing a unified, open-source framework that leverages symbolic mathematics, automatic regression, and dependency resolution to handle these disparate data sources. By combining a user-friendly YAML configuration with powerful backend processing, PyMatLib standardizes and simplifies the integration of realistic material behavior into scientific simulations.

Key Functionality

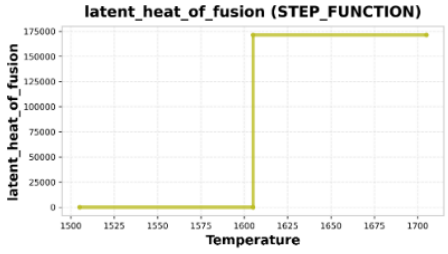
- **Flexible Input Methods**: The library supports six different property definition methods: constant values, step functions, file-based data (Excel, CSV, txt), tabular data, piecewise equations, and computed properties (Figure 1). This versatility allows users to leverage data from diverse sources, with robust file processing handled using pandas (McKinney, 2010).

```yaml
properties:
  density: 2700  # kg/m^3
```



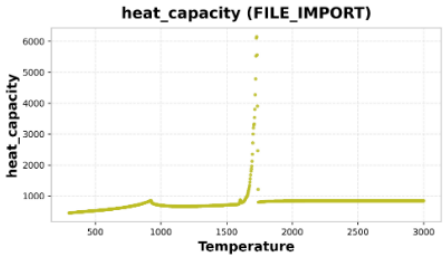
(a) Constant value

```yaml
properties:
  latent_heat_of_fusion:
    temperature: solidus_temperature
    value: [0.0, 171401.0]
```



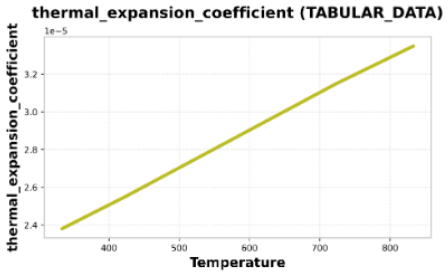
(b) Step function

```yaml
properties:
  heat_capacity:
    file_path: ./SS304L.xlsx
    temperature_column: T (K)
    property_column: Specific heat (J/(Kg
        K))
```



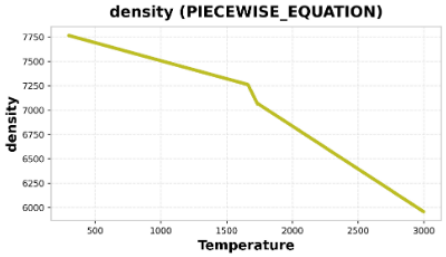
(c) File import

```yaml
properties:
  thermal_expansion_coefficient:
    temperature: [373.15, 473.15, 573.15,
        673.15, 773.15, 873.15]
    value: [24.56e-6, 26.54e-6, 28.51e-6,
        30.49e-6, 32.47e-6, 34.45e-6]
```



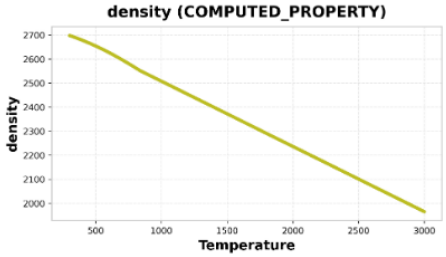
(d) Tabular data

```yaml
properties:
  density:
    temperature: [300, 1660, 1736, 3000]
    equation: [7877.39 - 0.37*T, 11816.63
        - 2.74*T, 8596.40 - 0.88*T]
```



(e) Piecewise equation

```yaml
properties:
  density:
    temperature: (300, 3000, 541)
    equation: 2700 / (1 + 3 *
        thermal_expansion_coefficient * (T
        - 293.15))
```



(f) Computed property

Figure 1: PyMatLib's property definition methods: (a) constant value, (b) step function, (c) file data, (d) tabular data, (e) piecewise equations, and (f) computed properties.

- **Universal Material Support**: The framework is designed with an extensible architecture to support any material type. It is currently implemented and thoroughly tested for pure metals and alloys through it's unified interface, with a modular design that allows for straightforward extension to other material classes such as ceramics, polymers, composites, or other specialized materials as research evolves.

- **Automatic Dependency Resolution**: For properties that depend on others (e.g., thermal diffusivity calculated from thermal conductivity, density, and heat capacity), PyMatLib automatically determines the correct processing order and resolves mathematical dependencies without manual intervention. The library detects circular dependencies and provides clear error messages for invalid configurations, freeing users from complex dependency management.
- **Regression and Data Reduction**: The library integrates pwlf (Jekel & Venter, 2019) to perform piecewise regression for large datasets. This simplifies complex property curves into efficient mathematical representations with configurable polynomial degrees and segments, reducing computational overhead while maintaining physical accuracy (Figure 2).
- **Configurable Boundary Behavior**: Users can define how properties behave outside their specified temperature ranges, choosing between constant-value or linear extrapolation to best match the physical behavior of the material. The boundary behavior options work seamlessly with the regression capabilities to provide comprehensive data processing control (Figure 2).

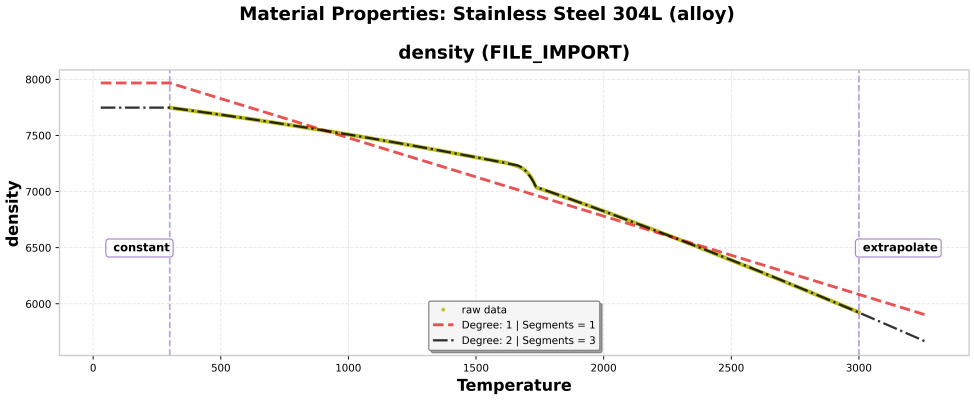


Figure 2: PyMatLib's data processing capabilities: regression and data reduction showing raw experimental data (points) fitted with different polynomial degrees and segment configurations, and boundary behavior options demonstrating constant versus extrapolate settings for the same density property, illustrating how PyMatLib can reduce complexity while maintaining physical accuracy and providing flexible boundary control.

- **Intelligent Simplification Timing**: PyMatLib provides sophisticated control over when data simplification occurs in the dependency chain via the `simplify` parameter. `simplify: pre` simplifies properties before they are used in dependent calculations, optimizing performance. With `simplify: post`, simplification is deferred until all dependent properties have been computed, maximizing numerical accuracy. This timing control allows users to balance computational efficiency with numerical accuracy based on their specific simulation requirements.

```yaml
bounds: [constant, extrapolate]  # Boundary behavior: 'constant' or 'extrapolate'
regression:      # Optional regression configuration
  simplify: pre  # 'pre' (before processing) or 'post' (after processing)
  degree: 2      # Polynomial degree for regression
  segments: 3    # Number of piecewise segments
```

- **Bidirectional Property-Variable Inversion**: The library can automatically generate inverse piecewise functions, enabling determination of independent variables from known property values (e.g., `temperature = f(property)`). This capability is essential for energy-

based numerical methods (Voller & Prakash, 1987), phase-change simulations, and iterative solvers. Currently focused on temperature-dependent properties, the underlying architecture supports future extension to additional variables such as concentration, pressure, or shear rate. Inverse function generation supports linear piecewise segments (either through default linear interpolation or explicit `degree=1` regression), ensuring robust mathematical invertibility.

- **Built-in Validation Framework**: A comprehensive validation framework checks YAML configurations for correctness, including composition sums, required fields for pure metals versus alloys, and valid property names. This prevents common configuration errors and ensures reproducible material definitions (Roache, 1998).
- **Integrated Visualization**: An integrated visualization tool using matplotlib (Hunter, 2007) allows users to automatically generate plots to verify their property definitions visually, with the option to disable visualization for production workflows after validation.

Usage

A material is defined in a YAML file and loaded with a single function call. The following examples demonstrate a pure metal and an alloy configuration, followed by the Python code to load and use the material.

YAML Configuration Examples

Pure Metal (`Al.yaml`)

```yaml
name: Aluminum
material_type: pure_metal

# Composition must sum to 1.0 (for pure metals, single element = 1.0)
composition:
  Al: 1.0  # Aluminum

# Required temperature properties for pure metals
melting_temperature: 933.47  # Solid becomes liquid (K)
boiling_temperature: 2743.0  # Liquid becomes gas (K)

properties:
  thermal_expansion_coefficient:
    temperature: [373.15, 473.15, 573.15, 673.15, 773.15, 873.15]
    value: [2.38e-05, 2.55e-05, 2.75e-05, 2.95e-05, 3.15e-05, 3.35e-05]
    bounds: [constant, constant]
    regression:
      simplify: post
      degree: 1
      segments: 1

  density:
    temperature: (300, 3000, 541)
    equation: 2700 / (1 + 3 * thermal_expansion_coefficient * (T - 293.15))
    bounds: [constant, constant]
    regression:
      simplify: pre
      degree: 1
      segments: 1
```

Alloy (`SS304L.yaml`)

```yaml
name: Stainless Steel 304L
material_type: alloy

# Composition fractions must sum to 1.0
composition:
  Fe: 0.675  # Iron
  Cr: 0.170  # Chromium
  Ni: 0.120  # Nickel
  Mo: 0.025  # Molybdenum
  Mn: 0.010  # Manganese

# Required temperature properties for alloys
solidus_temperature: 1605.          # Melting begins (K)
liquidus_temperature: 1735.         # Material is completely melted (K)
initial_boiling_temperature: 3090.  # Boiling begins (K)
final_boiling_temperature: 3200.    # Material is completely vaporized (K)

properties:
  density:
    file_path: ./SS304L.xlsx
    temperature_header: Temperature (K)
    value_header: Density (kg/(m)^3)
    bounds: [constant, extrapolate]
    regression:      # Optional regression configuration
      simplify: pre  # Simplify before processing
      degree: 2      # Use quadratic regression for simplification
      segments: 3    # Fit with 3 segments for piecewise linear approximation
```

Complete YAML configurations are provided in the PyMatLib documentation.

Python Integration

The primary entry point is the create_material function, which parses the YAML file and returns a fully configured material object.

```python
import sympy as sp
from pymatlib.parsing.api import create_material

# Create a material with a symbolic temperature variable
T = sp.Symbol('T')
aluminum = create_material('Al.yaml', T, enable_plotting=True)

# Access properties as symbolic expressions
print(f"Density: {aluminum.density}")
# Output: Piecewise((2678.43051234161, u_C < 300.0),
#                   (2744.36352618972 - 0.21977671282703*u_C, u_C < 3000.0),
#                   (2085.03338770863, True))

# Evaluate properties at a specific temperature
density_at_500K = aluminum.density.subs(T, 500).evalf()
print(f"Density at 500 K: {density_at_500K:.2f} kg/m^3")
# Output: Density at 500 K: 2634.48 kg/m^3
```

Comparison with Existing Tools

Feature	PyMatLib	CoolProp	NIST WebBook	CALPHAD
Core Capabilities				
Symbolic Integration	Yes	No	No	No
Dependency Resolution	Yes (Automatic)	No	No	No
Multiple Input Methods	Yes (6 types)	No	No	No
Material Support				
Solid Materials	Yes	Limited	Yes	Yes
Custom Properties	Yes (Any)	No	No	Limited
Temperature Dependencies	Yes	Yes	Yes	Yes
Accessibility				
Open Source	Yes	Yes	No	No
Python Integration	Native	Yes	API only	No

Key Advantage: PyMatLib's unique combination of native symbolic mathematics via SymPy (Meurer et al., 2017), automatic dependency resolution, and multiple input methods provides a level of flexibility and integration not found in existing tools, enabling more reproducible and sophisticated scientific simulations.

Research Applications and Availability

PyMatLib is applicable to a wide range of research areas, including alloy design and optimization (Callister & Rethwisch, 2018), finite element analysis (Hughes, 2012), multiscale modeling (Tadmor & Miller, 2011), computational fluid dynamics and heat transfer. Its architecture promotes reproducible science and is well-suited for high-performance computing environments, with demonstrated integrations into frameworks like pystencils (Bauer et al., 2019) and waLBerla (Bauer et al., 2021).

PyMatLib is an open-source software distributed under the BSD-3-Clause License. The source code, comprehensive documentation, and example configurations are available on GitHub.

Acknowledgements

The development of PyMatLib was supported by the Friedrich-Alexander-Universität Erlangen-Nürnberg. We acknowledge the developers of SymPy (Meurer et al., 2017), NumPy (Harris et al., 2020), pandas (McKinney, 2010), matplotlib (Hunter, 2007), and ruamel.yaml (Wel, 2023), whose libraries provide the symbolic mathematics, numerical computing, data processing, visualization, and configuration parsing capabilities that form the foundation of PyMatLib.

References

Ashby, M. F. (2013). *Materials and design: The art and science of material selection in product design* (3rd ed.). Butterworth-Heinemann. ISBN: 978-0080982052

Bauer, M., Hötzer, J., Ernst, D., Hammer, J., Seiz, M., Hierl, H., Hönig, J., Köstler, H., Nestler, B., & Rüde, U. (2019). Code generation for massively parallel phase-field simulations. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–12. https://doi.org/10.1145/3295500.3356186

Bauer, M., Köstler, H., & Rüde, U. (2021). waLBerla: A block-structured high-performance framework for multiphysics simulations. *Computers & Mathematics with Applications*, *81*, 478–501. https://doi.org/10.1016/j.camwa.2020.01.007

Bell, I. H., Wronski, J., Quoilin, S., & Lemort, V. (2014). Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library CoolProp. *Industrial & Engineering Chemistry Research*, *53*(6), 2498–2508. https://doi.org/10.1021/ie4033999

Callister, W. D., & Rethwisch, D. G. (2018). *Materials science and engineering: An introduction* (10th ed.). John Wiley & Sons. ISBN: 978-1119405498

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

Hughes, T. J. R. (2012). *The finite element method: Linear static and dynamic finite element analysis*. Dover Publications. ISBN: 978-0486411811

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Jekel, C. F., & Venter, G. (2019). *pwlf: A python library for fitting 1D continuous piecewise linear functions*. https://github.com/cjekel/piecewise_linear_fit_py

Lewis, R. W., Morgan, K., Thomas, H. R., & Seetharamu, K. N. (1996). *Finite element analysis of heat transfer and fluid flow*. John Wiley & Sons. ISBN: 978-0471943617

Lukas, H. L., Fries, S. G., & Sundman, B. (2007). *Computational thermodynamics: The calphad method*. Cambridge University Press. ISBN: 978-0521868112

McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

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

National Institute of Standards and Technology. (2023). *NIST chemistry WebBook*. https://webbook.nist.gov/chemistry/

Roache, P. J. (1998). *Verification and validation in computational science and engineering*. Hermosa Publishers. ISBN: 978-0913478080

Tadmor, E. B., & Miller, R. E. (2011). *Modeling materials: Continuum, atomistic and multiscale techniques*. Cambridge University Press. ISBN: 978-0521856980

Voller, V. R., & Prakash, C. (1987). A fixed grid numerical modelling methodology for convection-diffusion mushy region phase-change problems. *International Journal of Heat and Mass Transfer*, *30*(8), 1709–1719. https://doi.org/10.1016/0017-9310(87)90317-6

Wel, A. van der. (2023). *Ruamel.yaml: YAML 1.2 loader/dumper package for python*. https://yaml.readthedocs.io/

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L. B. da, Bourne, P. E., & others. (2016). The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 1–9. https://doi.org/10.1038/sdata.2016.18

Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2013). *The finite element method: Its basis and fundamentals* (7th ed.). Butterworth-Heinemann. ISBN: 978-1856176330