The Journal of Open Source Software

PyMatLib: A Python Library for Temperature-Dependent Material Property Processing in Scientific Simulations

Rahil Miten Doshi [1]

1 Friedrich-Alexander-Universität Erlangen-Nürnberg, 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Published: unpublished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 temperature-dependent material properties in computational simulations. The physical characteristics of materials, such as thermal conductivity, density, and heat capacity, in fields like metal casting, heat treatment, and thermal analysis, change significantly with temperature. Accurately modeling these changes is a persistent challenge.

PyMatLib addresses this by allowing researchers to define complex material behaviors in simple human-readable YAML files, which 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 in materials science and heat transfer, and serves as a seamless bridge between experimental data and numerical simulation.

Statement of Need

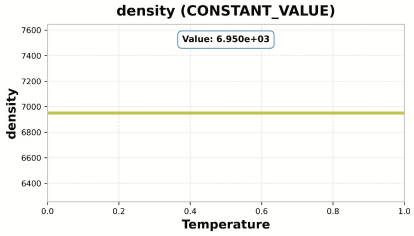
The accurate representation of temperature-dependent material properties is fundamental to the fidelity of computational materials science, thermal analysis, and multi-physics simulations (Lewis et al., 1996; Zienkiewicz et al., 2013). Researchers often rely on manual interpolation, custom scripting, or proprietary software solutions, which can lead to issues with reproducibility, flexibility and standardization (Ashby, 2013). While valuable resources like NIST (National Institute of Standards and Technology, 2023) and libraries like CoolProp (Bell et al., 2014) provide extensive material data, they primarily offer raw tabular data or focus on fluid properties, lacking the integrated symbolic processing and dependency management capabilities needed for complex simulations. Similarly, specialized CALPHAD databases (Lukas et al., 2007) require proprietary software and do not easily integrate with general-purpose simulation codes.

This gap forces researchers to develop ad-hoc solutions for each new project, creating a bottleneck in the research workflow and hindering the FAIR principles of data sharing (Wilkinson et al., 2016). PyMatLib was created to fill this gap by providing a robust, flexible, and open-source tool that combines a user-friendly configuration system with powerful backend processing, thereby standardizing and simplifying the integration of realistic material behavior into scientific simulations.

Key Functionality

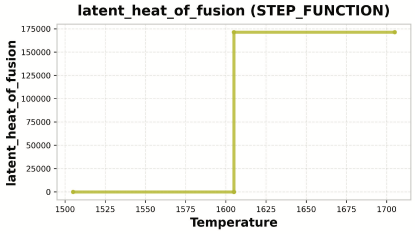
- **Flexible Input Methods**: The library supports six different property definition methods: constant values, step functions, file-based data (Excel, CSV, txt), tabular data, piecewise equations, and computed properties (Figure 1). This versatility allows users to leverage data from diverse sources,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

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



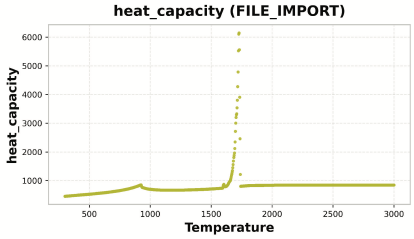
(a) Constant value

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



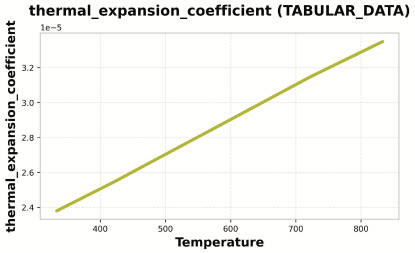
(b) Step function

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



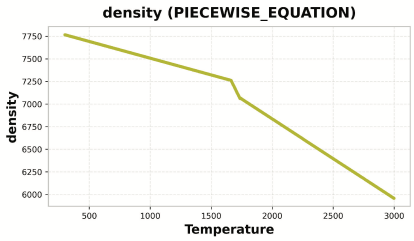
(c) File import

```yaml
properties:
  thermal_expansion_coefficient:
    temperature: [333.15, 423.15, 523.15,
                  623.15, 723.15, 833.15]
    value: [2.38e-05, 2.55e-05, 2.75e-05,
```



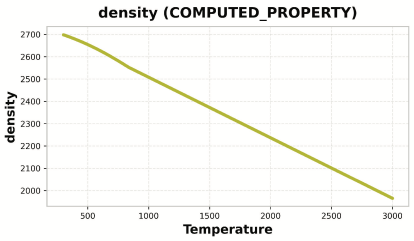
(d) Tabular data

```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
```



(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**: A unified interface supports both pure metals and alloys, with

extensibility for additional material types. Pure metals use melting/boiling temperatures, while alloys use solidus/liquidus temperature ranges.

- **Automatic Dependency Resolution**: For properties that depend on others (e.g., thermal diffusivity), PyMatLib automatically determines the correct calculation order and detects circular dependencies, freeing the user from manual implementation.
- **Intelligent Simplification Timing**: PyMatLib provides sophisticated control over when data simplification occurs in the dependency chain through the `simplify` parameter. With `simplify: pre`, properties are simplified using regression before being 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.
- **Regression and Data Reduction**: The library integrates pwlf (Jekel & Venter, 2019) to perform piecewise linear regression for large datasets. This simplifies complex property curves into efficient, accurate mathematical representations with configurable polynomial degrees and segments, reducing computational overhead while maintaining physical accuracy (Figure 2).

```yaml
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
```

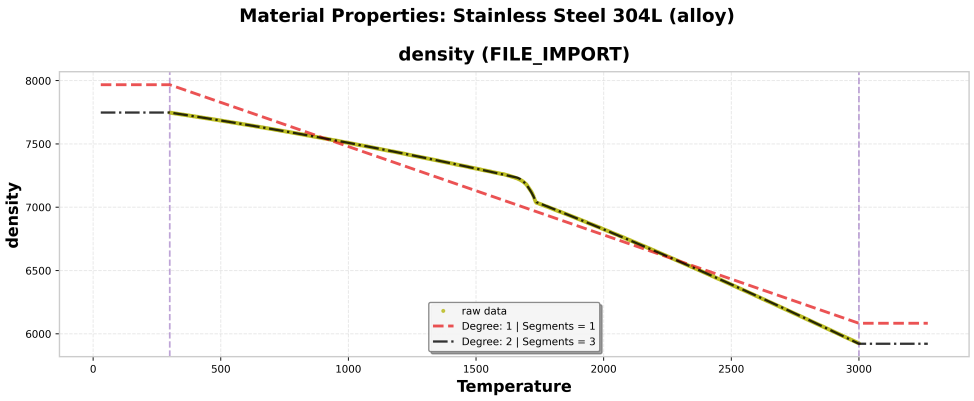


Figure 2: Regression capabilities showing data simplification effects: raw experimental data (points) fitted with different polynomial degrees and segment configurations, demonstrating how PyMatLib can reduce complexity while maintaining physical accuracy.

- **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 (Figure 3).

```yaml
bounds: [constant, extrapolate]  # Boundary behavior: 'constant' or 'extrapolate'
```

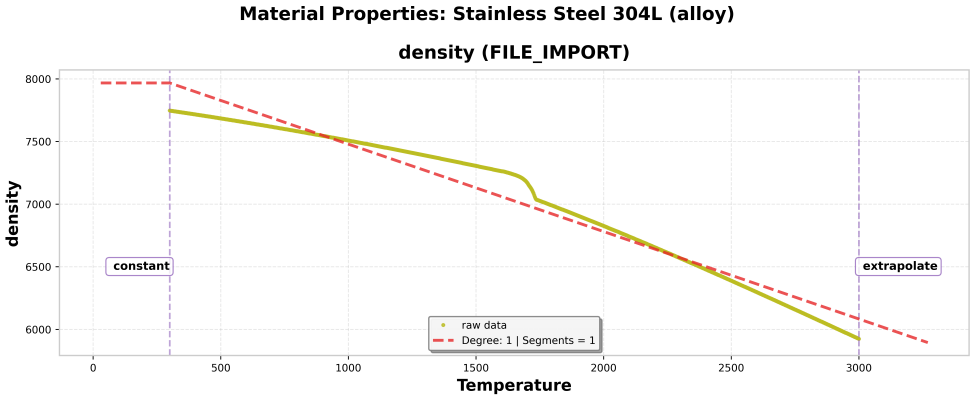


Figure 3: Boundary behavior options in PyMatLib showing the same density property with different extrapolation settings: constant boundaries (left) maintain edge values outside the defined range, while extrapolate boundaries (right) use linear extrapolation.

- **Automatic Dependency Resolution**: Intelligent processing order determination for computed properties ensures mathematical dependencies are resolved correctly without manual intervention. The library automatically detects circular dependencies and provides clear error messages for invalid configurations.
- **Bidirectional Property-Temperature Conversion**: The library can automatically generate inverse piecewise functions (`temperature = f(property)`), a critical feature for energy-based numerical methods (Voller & Prakash, 1987), phase-change simulations, and iterative solvers where `temperature` is the unknown variable. The 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 Example

PyMatLib is designed for ease of use. A material is defined in a YAML file and loaded with a single function call. The YAML files can include pure metals with melting/boiling temperatures or alloys with solidus/liquidus temperature ranges.

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
```

```yaml
# 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: [333.15, 423.15, 523.15, 623.15, 723.15, 833.15]  # Explicit temperatur
    value: [2.38e-05, 2.55e-05, 2.75e-05, 2.95e-05, 3.15e-05, 3.35e-05]  # 1/K values
    bounds: [constant, constant]
    regression:
      simplify: post
      degree: 1
      segments: 1

  density:
    temperature: (300, 3000, 541)
    equation: 2700 * (1 - 3*thermal_expansion_coefficient * (T - 293))
    bounds: [constant, constant]
```

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.         # Melting 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: T (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 for both are provided in the PyMatLib documentation.

Python Usage:

Integrating PyMatLib into a scientific workflow is straightforward. 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}")

# Evaluate properties at a specific temperature
density_at_300K = aluminum.density.subs(T, 300).evalf()
print(f"Density at 300 K: {density_at_300K:.2f} 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	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), energy-based finite element methods for thermal analysis (Hughes, 2012), multiscale simulations (Tadmor & Miller, 2011), and high-performance computing in 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 open-source under the BSD-3-Clause license. The source code, documentation, and further examples are available on GitHub..

The Journal of Open Source Software

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