The Journal of Open Source Software

# MaterForge: Materials Formulation Engine with Python

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

**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:** 10.21105/joss.09909

**Software**

- Review
- Repository
- Archive

**Editor:** Sarath Menon

**Reviewers:**

- @jan-janssen
- @JosePizarro3

**Submitted:** 20 August 2025
**Published:** 13 May 2026



## Summary

MaterForge is an extensible, open-source Python library that streamlines the definition and use of material properties in numerical simulations. The library supports complex material behaviors, from simple constants to experimental data, in user-friendly YAML configurations. These are internally converted into symbolic mathematical expressions for scientific computing frameworks. MaterForge supports any material type through a schema-agnostic design, provides flexible property definitions, and automatically resolves dependency order for derived properties while detecting cycles. It is designed for high-performance computing (HPC) applications and serves as a bridge between experimental data and numerical simulation.

## Statement of Need

Accurate numerical simulations of physical processes rely on well-characterized material properties - quantities such as thermal conductivity, density, and viscosity that depend on state variables like temperature, pressure, or strain rate (Lewis et al., 1996). This challenge is compounded by the wide variation in data availability, from well-characterized models for established materials to sparse experimental points for novel materials. Property definitions consequently range from simple constants to complex tabular datasets or sophisticated equations, creating significant integration hurdles for researchers.

To manage this complexity, researchers often resort to manual interpolation, custom scripting, or proprietary software, which compromises reproducibility and standardization (Ashby & Johnson, 2014). While valuable resources like the NIST WebBook (Linstrom & Mallard, 2001) and CoolProp (Bell et al., 2014) provide raw data, they lack integrated processing to unify these varied formats into simulation-ready symbolic expressions. Thermodynamic modeling tools such as pycalphad (Otis & Liu, 2017) and CALPHAD databases (Lukas et al., 2007) are powerful for phase equilibria calculations but operate at a different layer of the workflow: they generate property data, not simulation-ready expressions.

This creates a gap between property data generation and simulation integration, leading to ad hoc solutions that hinder workflow efficiency and FAIR data adoption (Wilkinson et al., 2016). MaterForge fills this gap by occupying a dedicated post-processing layer in the materials simulation workflow, complementing rather than competing with thermodynamic modeling tools: pycalphad generates the data; MaterForge prepares it for simulation.

## Simulation Workflow

MaterForge is designed to operate as the intermediate layer in a three-stage workflow:

**Stage 1 - Property Data Generation**: Raw material property data is produced by upstream tools such as pycalphad (Otis & Liu, 2017) for thermodynamic phase equilibria, CoolProp (Bell et al., 2014) or the NIST WebBook (Linstrom & Mallard, 2001) for fluid properties, or commercial tools such as JMatPro and Thermo-Calc for alloy properties, or direct experimental measurement. These tools generate tabular data, equations, or database entries.

**Stage 2 - MaterForge Post-Processing**: MaterForge ingests this data via YAML configuration files and converts it into optimized symbolic mathematical expressions using SymPy (Meurer et al., 2017). It performs automatic regression and data reduction, resolves inter-property dependencies, validates configurations, and generates visualization plots for verification. The output is a fully configured `Material` object in which each property is stored as a symbolic expression - a callable function of a chosen dependency variable such as temperature, pressure, or composition - ready for direct use in simulation codes.

**Stage 3 - Simulation Integration**: The symbolic expressions are passed directly into simulation frameworks such as pystencils (Bauer et al., 2019) and waLBerla (Bauer et al., 2021) for code generation, or into any Python-based CFD solver. Because properties are SymPy expressions, they plug into symbolic assignment collections without any additional conversion.

## Key Functionality

- **Flexible Input Methods**: The library supports various property definition methods such as constant values, step functions, file-based data (.xlsx, .csv, .txt), tabular data, piecewise equations, and computed properties. The example configuration in subsection 5.1 combines these input types in a single material, and the resulting property curves are visualized in Figure 2.
- **Schema-Agnostic Material Support**: The framework imposes no structural constraints on material definitions. Any material kind - pure metals, alloys, ceramics, polymers, composites, or hypothetical materials - and any property name are valid. The only required YAML fields are `name` and `properties`. This design has been validated across steel alloys, aluminum, and Al2O3 ceramic configurations.
- **Automatic Dependency Resolution**: For dependent properties (e.g., thermal diffusivity calculated from thermal conductivity, density, and heat capacity), MaterForge automatically determines the correct processing order, resolves mathematical dependencies, and detects circular references.
- **Regression and Data Reduction**: The library performs piecewise regression for large datasets, simplifying complex property curves into efficient mathematical representations with configurable polynomial degrees and segment counts, reducing computational overhead while maintaining accuracy.
- **Intelligent Simplification Timing**: MaterForge provides sophisticated control over when data simplification occurs via the `simplify` parameter. `simplify: pre` optimizes performance by simplifying properties before they are used in dependent calculations, while `simplify: post` defers simplification until all dependent properties have been computed, maximizing numerical accuracy.
- **Configurable Boundary Behavior**: Users can define how properties behave outside their specified ranges, choosing between `constant`-value clamping 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 1).

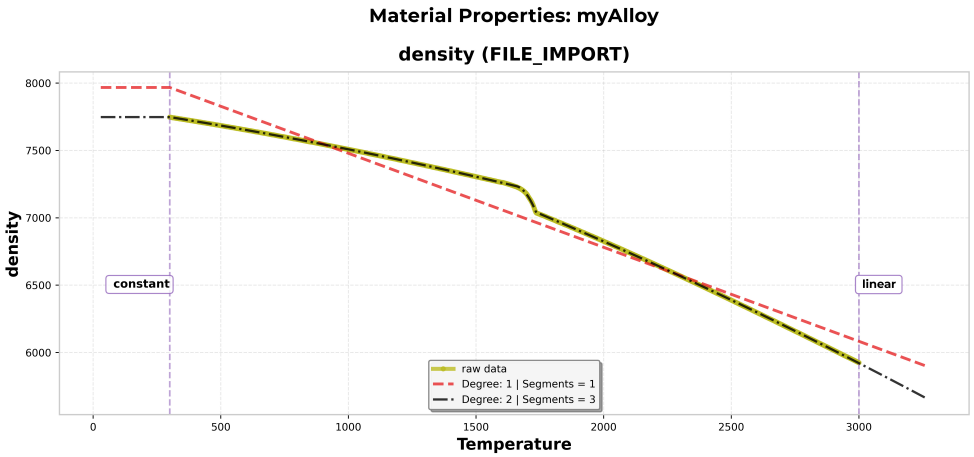


**Figure 1:** MaterForge's data processing capabilities: regression and data reduction showing raw data (green) fitted with different polynomial degrees and segment configurations, and configurable boundary behavior options demonstrating constant versus linear extrapolation for the same density property, illustrating how MaterForge reduces complexity while maintaining physical accuracy.

- **Inverse Property Computation**: The library can generate inverse piecewise-linear functions, enabling the determination of the independent variable from a known property value. This capability is essential for energy-based numerical methods (Voller & Prakash, 1987), where temperature is recovered via the inverse of the specific enthalpy function.
- **Built-in Validation Framework**: A comprehensive validation framework checks YAML configurations for correctness, including structural validation, required fields, property type detection, and dependency cycle detection, preventing common configuration errors before simulation begins (Roache, 1998).
- **Integrated Visualization**: An integrated visualization tool automatically generates plots to verify property definitions, with the option to disable visualization for production workflows.

## Usage

Materials are defined in YAML files and loaded via `create_material`, which returns a fully configured `Material` object. All material properties live in the `properties` block - the only other required top-level field is `name`. Named properties can be referenced in other property configurations: scalar constants are valid anywhere, while full expressions are valid in `COMPUTED_PROPERTY` equations only. MaterForge automatically resolves the correct evaluation order. The example `myAlloy` configuration below combines several input methods and is used to generate the plots shown in Figure 2.

### YAML Configuration Example: `myAlloy.yaml`

```yaml
name: myAlloy

properties:

    density: 6950  # Fig. 2a

    latent_heat_of_fusion:  # Fig. 2b
        dependency: density / 4.32
```

```yaml
        value: [0, 171401]
        bounds: [constant, constant]

    heat_capacity:  # Fig. 2c
        file_path: ./myAlloy.csv
        dependency_column: T (K)
        property_column: Specific heat (J/(Kg K))
        bounds: [constant, constant]
        regression:
            simplify: pre
            degree: 3
            segments: 6

    heat_conductivity:  # Fig. 2d
        dependency: [500, 1000, 1600, 1700, 1750, 2000, 2500]
        value: [19.25, 25.47, 32.94, 33.52, 31.53, 35.33, 42.95]
        bounds: [linear, linear]

    viscosity:  # Fig. 2e
        dependency: [300, 1660, 1736, 3000]
        equation: [7877.39-0.37*T, 11816.63-2.74*T, 8596.40-0.88*T]
        bounds: [constant, constant]

    thermal_diffusivity:  # Fig. 2f
        dependency: (3000, 300, -5.0)
        equation: heat_conductivity / (density * heat_capacity)
        bounds: [constant, linear]
        regression:
            simplify: post
            degree: 3
            segments: 7
```

## Python Integration

```python
import sympy as sp
from materforge import create_material


# Define the dependency variable and load material from YAML
T = sp.Symbol('T')
myAlloy = create_material('myAlloy.yaml', T, enable_plotting=True)


# Access a symbolic property expression - a SymPy Piecewise function of T
density_expr = myAlloy.density


# Evaluate all properties at 500 K - returns a new Material with numeric values
myAlloy_at_500K = myAlloy.evaluate(T, 500.0)
print(float(myAlloy_at_500K.density))   # numeric density at 500 K
```

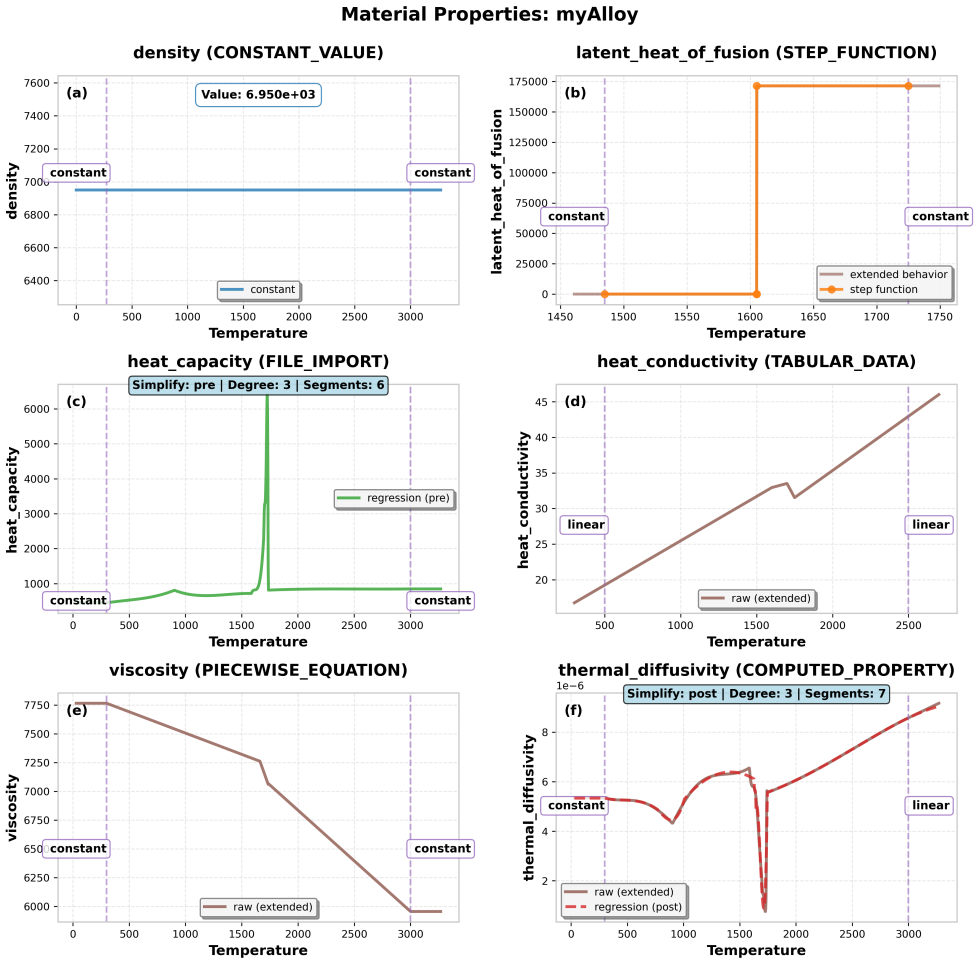


**Figure 2:** Automatically generated material property plots for the example alloy `myAlloy` defined in subsection 5.1, illustrating constant (a), step-function (b), file-based (c), tabular (d), piecewise-equation (e), and computed (f) properties.

## Research Applications

MaterForge is applicable to alloy design (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 HPC environments, with demonstrated integrations into frameworks like pystencils (Bauer et al., 2019) and waLBerla (Bauer et al., 2021).

## Availability

MaterForge is distributed under the BSD-3-Clause License. The source code is hosted on GitHub, with full documentation and YAML examples. The package can be installed via PyPI using `pip install materforge`.

## AI Usage Disclosure

GitHub Copilot and Claude Sonnet assisted with boilerplate code, refactoring, and text editing in the manuscript and documentation. All AI-assisted outputs were reviewed, edited, and validated by the human authors, who designed the overall code architecture and take full responsibility for the accuracy and originality of the software and all submitted materials.

## Acknowledgements

This work was funded by the European High Performance Computing Joint Undertaking (Grant No. 101093457) and the Deutsche Forschungsgemeinschaft within Research Unit FOR-5134 (Grant No. 434946896).

## References

Ashby, M., & Johnson, K. (Eds.). (2014). Materials and design. In *Materials and design (third edition)* (Third Edition, p. i). Butterworth-Heinemann. https://doi.org/10.1016/B978-0-08-098205-2.00011-1

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

Hughes, T. J. R. (2012). *The finite element method: Linear static and dynamic finite element analysis*. Dover Publications. https://doi.org/10.1016/0045-7825(87)90013-2

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

Linstrom, P. J., & Mallard, W. G. (2001). The NIST chemistry WebBook: A chemical data resource on the internet. *Journal of Chemical & Engineering Data*, *46*(5), 1059–1063. https://doi.org/10.1021/je000236i

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

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

Otis, R., & Liu, Z.-K. (2017). Pycalphad: CALPHAD-based computational thermodynamics in python. *Journal of Open Research Software*, *5*(1), 1. https://doi.org/10.5334/jors.140

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 modeling 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

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