The Journal of Open Source Software

LowLevelFEM.jl: A lightweight finite element toolbox in Julia

Balázs Pere [1]

1 Department of Applied Mechanics, Széchenyi István University, Győr, Hungary

DOI: 10.21105/joss.10096

Software

- Review
- Repository
- Archive

Editor: Prashant Jha

Reviewers:

- @cfarm6
- @TimJosephson

Submitted: 07 September 2025
Published: 28 July 2026

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

Summary

LowLevelFEM.jl is a finite element method (FEM) (Zienkiewicz & Taylor, 2005) toolbox whose core finite element algorithms are implemented in the Julia programming language (Bezanson et al., 2017). It integrates with Gmsh for geometry creation, meshing, and visualization. Its design philosophy emphasizes **simplicity, transparency, and operator-based weak-form formulation and prototyping**, making it suitable for both educational and research purposes in mechanics and engineering. Unlike frameworks that primarily expose high-level domain-specific languages (DSLs), LowLevelFEM provides direct access to the underlying FEM building blocks in Julia, enabling full control over discretization, assembly, and solution steps, while also supporting an operator-based DSL for expressing custom weak-form formulations.

The package currently supports two- and three-dimensional solid mechanics problems including plane stress, plane strain, axisymmetric, and 3D solid analyses. Its minimal design lowers the entry barrier for students while still offering enough flexibility for advanced users to **inspect and control algorithms step by step, process intermediate results during the solution procedure, and perform operations on scalar, vector, and tensor fields**. LowLevelFEM uses **Gmsh** (Geuzaine & Remacle, 2009) as its pre- and post-processor, ensuring compatibility with a widely adopted meshing and visualization tool. The toolbox also supports geometrically nonlinear formulations based on a Total Lagrangian framework and energy-based constitutive modeling, enabling research-level investigations beyond linear elasticity.

Thanks to Julia's just-in-time compilation and multiple dispatch, the implementation remains concise while preserving transparency and extensibility for educational use and rapid research prototyping. LowLevelFEM is released under the MIT license and distributed through the Julia General registry. The online documentation, including a growing collection of executable Jupyter notebook tutorials, is available at https://perebalazs.github.io/LowLevelFEM.jl/stable/

Example

Below is a simple example illustrating a typical LowLevelFEM workflow using Gmsh for pre- and post-processing:

```
using LowLevelFEM

openGeometry("model.geo")

mat  = Material("body", E=2e5, ν=0.3)
prob = Problem([mat], type=:PlaneStrain)
# Other problem types: :Solid, :PlaneStress, :AxiSymmetric,
# :HeatConduction, :PlaneHeatConduction, etc.

bc   = displacementConstraint("supp", ux=0, uy=0)
```

```
force = load("load", fy=-1)

u = solveDisplacement(prob, load=[force], support=[bc])
S = solveStress(u)

showDoFResults(u)
showDoFResults(u, :ux)
showStressResults(S)
showStressResults(S, :sxy, name="Shear stress")

openPostProcessor()
```

Note: physical group names in the geometry (created in Gmsh) must match the strings used above (e.g., “body”, “supp”, “load”).

Alternatively, a lower-level sequence:

```
K = stiffnessMatrix(prob)
f = loadVector(prob, [force])
u = solveDisplacement(K, f, support=[bc])

# Simple Hooke's law stress computation
E = mat.E
ν = mat.ν
# Expand the 2D displacement field for tensor-based stress computation
u = expandTo3D(u)
A = (u ∘ ∇ + ∇ ∘ u) / 2
I = TensorField(prob, "body", [1 0 0; 0 1 0; 0 0 1])
S = E / (1 + ν) * (A + ν / (1 - 2ν) * trace(A) * I)
```

The project repository includes a growing collection of executable Jupyter notebook tutorials that demonstrate both standard workflows and custom weak-form formulations.

Statement of need

Finite element simulations are essential in many fields of engineering, especially in solid mechanics and structural analysis. However, educational and research communities often face two challenges:

1. **Accessibility**: Commercial FEM packages are expensive and closed-source, limiting their use in academic teaching and reproducible research.
2. **Extensibility**: Large open-source frameworks such as FEniCS (Logg et al., 2012) and deal.II (Bangerth et al., 2007) provide comprehensive finite element environments. In contrast, LowLevelFEM is designed to expose the underlying finite element algorithms directly in Julia, facilitating rapid prototyping of new formulations and operators.

LowLevelFEM addresses these challenges by offering a lightweight implementation whose core finite element routines are written in Julia while relying on Gmsh for geometry creation, meshing, and visualization. This makes the package particularly well-suited for:

- Teaching FEM concepts in undergraduate and graduate courses.
- Rapid prototyping of custom finite element formulations, weak-form operators, and **custom solution algorithms**.
- Research projects where step-by-step inspection of the solution process and manipulation of intermediate fields is required.
- Development, testing, and validation of novel finite element formulations.

By combining transparent algorithms with Julia's scientific ecosystem (e.g., LinearAlgebra.jl, Plots.jl) and by relying on **Gmsh** (Geuzaine & Remacle, 2009) for pre- and post-processing, LowLevelFEM serves as a bridge between pedagogy and advanced research workflows. It also complements existing Julia FEM frameworks such as Gridap.jl (Badia et al., 2020) and interfaces naturally with Arpack.jl (JuliaLinearAlgebra, 2026) for sparse eigenvalue problems, enabling modal analysis and linear buckling calculations.

State of the field

Several finite element frameworks are available in the Julia ecosystem and beyond. High-level Julia packages such as Gridap.jl (Badia et al., 2020) provide domain-specific abstractions for variational formulations, while Ferrite.jl (Carlsson et al., 2021), building upon the earlier JuAFEM.jl (Carlsson, 2018), focuses on flexible but more structured implementations. Outside the Julia ecosystem, mature frameworks such as FEniCS (Logg et al., 2012) and deal.II (Bangerth et al., 2007) address similar application domains through different software architectures.

LowLevelFEM differs from these frameworks in its explicit operator-level design philosophy. Instead of abstracting away the assembly process through domain-specific languages or symbolic formulations, it exposes matrix assembly, intermediate scalar/vector/tensor fields, differential operators, and weak-form composition directly in Julia code. This makes it particularly suitable for educational purposes and for researchers who need full algorithmic transparency and control over intermediate computational steps.

Software design

LowLevelFEM is structured around a minimal set of core abstractions. The `Problem` type encapsulates material definitions, physical groups imported from Gmsh, and analysis settings (e.g., plane stress, 3D solid, heat conduction). Assembly routines such as `stiffnessMatrix`, `loadVector`, and `applyBoundaryConditions!` operate directly on these problem definitions.

The package separates:

- mesh and geometry handling (delegated to Gmsh),
- operator assembly,
- solution procedures,
- post-processing of scalar, vector, and tensor fields.

This modular structure allows users to either follow a high-level workflow (`solveDisplacement`, `solveStress`) or construct custom pipelines at a lower level. The implementation leverages Julia's multiple dispatch and just-in-time compilation to maintain readable code while providing efficient execution for educational and research applications.

In addition to linear formulations, the software includes a Total Lagrangian nonlinear pipeline based on strain energy functions. Constitutive models can be defined through free energy densities, from which stress measures and consistent tangent operators are computed through the implemented energy-based formulation. This operator-oriented design makes it possible to experiment with alternative bilinear and nonlinear forms without modifying compiled backends.

The package provides algebraic and differential operations on scalar, vector, and tensor fields, allowing users to compose continuum-mechanics expressions directly in Julia syntax, including, for example, gradient, divergence, trace, and tensor-contraction operations defined at the discrete level. These operators can be composed to express weak-form formulations directly in

Julia, as demonstrated in the *"Multifield Weak-Form DSL – Navier–Stokes"* tutorial, bridging the gap between mathematical notation and executable Julia code while keeping intermediate quantities explicit and user-accessible. This capability is particularly valuable for educational demonstrations and rapid prototyping of new finite element formulations.

Research impact statement

LowLevelFEM supports both educational and research activities in computational mechanics. In teaching, it enables students to inspect finite element algorithms step by step without switching languages or interacting with opaque compiled backends. In research, it facilitates rapid prototyping of constitutive models, custom operators, and weak-form formulations, allowing new finite element methods to be expressed in a notation close to their mathematical representation.

The package has been used in undergraduate and graduate finite element courses at Széchenyi István University, where it supports hands-on demonstrations and executable Jupyter notebook tutorials, and serves as a foundation for ongoing developments in nonlinear and multi-field finite element formulations. Its open-source MIT license and integration with the Julia ecosystem promote reproducible research and extensibility.

The package currently serves as the basis for ongoing research on nonlinear elasticity, multi-field finite element formulations, and operator-based finite element methods. Its executable Jupyter notebook tutorials also make it suitable for classroom demonstrations, allowing students to inspect every stage of the finite element workflow from weak-form definition to post-processing.

AI usage disclosure

Generative AI tools were used for minor language editing and documentation refinement. All scientific concepts, algorithms, and software implementations were designed, developed, and verified by the author.

References

Badia, S., Martín, A., & Verdugo, F. (2020). Gridap: An extensible finite element toolbox in Julia. *Journal of Open Source Software*, *5*(52), 2520. https://doi.org/10.21105/joss.02520

Bangerth, W., Hartmann, R., & Kanschat, G. (2007). deal.II–a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, *33*(4), 24. https://doi.org/10.1145/1268776.1268779

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Carlsson, K. (2018). *JuAFEM.jl*. https://github.com/KristofferC/JuAFEM.jl.

Carlsson, K., Ekre, F., & Contributors. (2021). *Ferrite.jl*. https://doi.org/10.5281/zenodo.13862652

Geuzaine, C., & Remacle, J.-F. (2009). Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*(11), 1309–1331. https://doi.org/10.1002/nme.2579

JuliaLinearAlgebra. (2026). *Arpack.jl*. https://github.com/JuliaLinearAlgebra/Arpack.jl.

Logg, A., Mardal, K.-A., & Wells, G. (2012). *Automated solution of differential equations by the finite element method: The FEniCS book*. Springer. https://doi.org/10.1007/978-3-642-23099-8

Zienkiewicz, O. C., & Taylor, R. L. (2005). *The finite element method* (6th ed.). Butterworth–Heinemann. ISBN: 9780080472775