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.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 07 September 2025
Published: unpublished

Summary

LowLevelFEM.jl is a finite element method (FEM) (Zienkiewicz & Taylor, 2005) toolbox written entirely in the Julia programming language (Bezanson et al., 2017). Its design philosophy emphasizes **simplicity, transparency, and performance**, making it suitable for both educational and research purposes in mechanics and engineering. Unlike large frameworks that rely on domain-specific languages or compiled backends, LowLevelFEM provides users with direct access to FEM building blocks in Julia, enabling full control over discretization, assembly, and solution steps.

The package currently supports two- and three-dimensional solid mechanics problems including plane stress, plane strain, axisymmetric, and 3D solid analyses. Its minimalistic 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 with 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.

Thanks to Julia's just-in-time compilation and multiple dispatch, the code remains concise while achieving performance comparable to traditional FEM codes in C/C++ or Fortran. LowLevelFEM is released under the MIT license and distributed via the Julia General registry. Documentation, tutorials, and examples are available at https://juliahub.com/ui/Packages/General/LowLevelFEM or 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:

```julia
using LowLevelFEM

# `gmsh` is exported by LowLevelFEM
gmsh.initialize()
gmsh.open("model.geo")

mat = material("body", E=2e5, ν=0.3)
prob = Problem([mat], type=:PlaneStress)  # :Solid, :PlaneStrain, :AxiSymmetric, :HeatCo

bc   = displacementConstraint("supp", ux=0, uy=0)
force = load("load", fy=-1)

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

The Journal of Open Source Software

```
showDoFResults(u)
showStressResults(S)

openPostProcessor()
gmsh.finalize()
```

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])
applyBoundaryConditions!(K, f, [bc])
u = K \ f

# Simple Hooke's law stress computation
A = (u ∘ ∇ + ∇ ∘ u) / 2
I = unitTensor(A)
S = E / (1 + ν) * (A + ν / (1 - 2 ν) * trace(A) * I)
```

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) or deal.II (Bangerth et al., 2015) provide powerful high-level interfaces but are difficult to extend at the low-level assembly stage without diving into C++ backends.

LowLevelFEM addresses these challenges by offering a **lightweight Julia-only implementation** that exposes all the core FEM routines directly in the high-level language. This makes the package particularly well-suited for:

- Teaching FEM concepts in undergraduate and graduate courses.
- Rapid prototyping of new FEM formulations and **non-standard algorithms**.
- Research projects where step-by-step inspection of the solution process and manipulation of intermediate fields is required.
- Demonstrations of Julia's potential as a performant and expressive language for numerical mechanics (Bezanson et al., 2017).

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 linear algebra tools like Arpack.jl (Knyazev, 2017).

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., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., Maier, M., Turcksin, B., & Young, D. (2015). The deal.II library, version 8.2. *Archive of Numerical Software*, *3*(100), 1–8. https://doi.org/10.11588/ans.2015.100.20553

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

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

Knyazev, A. (2017). Arpack.jl: A julia interface to ARPACK. *Journal of Open Source Software*, *2*(12), 142. https://doi.org/10.21105/joss.00142

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.