The Journal of Open Source Software

FEM_2D: A Rust Package for 2D Finite Element Method Computations with Extensive Support for *hp*-refinement

Jeremiah Corrado[1], **Jake J. Harmon**[1], **Milan M. Ilic**[1, 2], **and Branislav M. Notaroš**[1]

1 Colorado State University; Department of Electrical and Computer Engineering **2** University of Belgrade; School of Electrical Engineering

DOI: 10.21105/joss.04172

Software

- Review
- Repository
- Archive

Editor: Jed Brown

Reviewers:

- @jeremylt
- @YohannDudouit

Submitted: 13 February 2022
Published: 16 February 2022

Summary

The Finite Element Method (FEM) is a powerful tool used to solve Partial Differential Equations (PDE)s on arbitrary geometries. As its name suggests, the method works by breaking a geometric model (a Domain) into a set of small elements, each of which assigned a set of Basis Functions. A solution is expressed in terms of a weighted superposition of all the functions in the Domain such that the PDE is satisfied along with some continuity and boundary conditions. Some common applications include the Navier-Stokes equations which characterize the behavior of fluids, Schrödinger's equation which governs the evolution of quantum systems, and Maxwell's Equations which are a macroscopic description of essentially all Electromagnetic phenomena.

The `FEM_2D` library is a powerful FEM simulation tool implemented in Rust. It was designed within the domain of Computational Electromagnetics (CEM) with a specific focus on solving Maxwell's Equations over a 2D waveguide cross-sections with very high accuracy. This problem has practical engineering applications for the design of waveguides; however, it is used here as a research tool for exploring and evaluating improvements upon FEM.

Although its initial use case was domain specific, `FEM_2D`'s functionality extends easily to other domains using an API based on Rust `Traits` (which are like C++20 `Concepts` or Java `Interfaces`). Traits are a highly expressive form of genericism which allow functions to act on any data structure so long as it implements some shared functionality. In the case of solving PDEs with FEM, that generic functionality is the ability to compute an integral over an expression of two overlapping basis functions. As such, the library's entire functionality can be leveraged against a given PDE problem by implementing FEM_2D's `Integral` trait to express the variational form of the problem of interest.

`FEM_2D` has all the functionality needed to formulate and solve Generalized Eigenvalue Problems. This includes an easy-to-use Mesh API which handles Mesh instantiation and refinement. It also includes a Domain API which defines a Generic set of Basis Functions over the Mesh while ensuring adherence to continuity and boundary conditions. Additionally, there are two Eigenvalue-Problem solvers. The first is implemented using Nalgebra ("Nalgebra," 2021): a powerful linear algebra library written in Rust. The second is an external sparse solver –for large or poorly conditioned problems– implemented using SLEPc (Hernandez et al., 2005) (Balay et al., 2021b) (Balay et al., 2021a) (Balay et al., 1997). The library also has extensive functionality for generating `.vtk` files of solutions which can be plotted using external tools.

The Journal of Open Source Software

Statement of Need

FEM_2D's primary advantage over other FEM libraries, such as the Deal.II library (Arndt et al., 2021), is its highly dynamic and expressive *hp*-refinement API. Unlike many other quadrilateral-element FEM packages, `FEM_2D` supports n-irregular anisotropic *h*-refinement as well as anisotropic *p*-refinement. In other words, there are far fewer limitations on the shape, location, or orientation of new elements when adding them to the Mesh. The polynomial expansion orders of the Basis Functions associated with each element can also be modified separately in each direction.

Efficiently computing solutions over geometries with sharp edges or stark material discontinuities necessitates *hp*-refinement (whether isotropic or anisotropic). These situations tend to introduce multi-scale solution behavior which is challenging to model with pure *p*- or pure *h*-refinements, motivating combined hp-refinements (Harmon et al., 2021). Within the class of hp-refinements, the addition of anisotropic *hp*-refinements (over isotropic ones) presents a significantly larger capacity for solution efficiency, as small-scale behavior is targeted more directly and ineffectual Degrees of Freedom are left out of the system (Corrado et al., 2021).

The theory behind this library's *h*-refinement methodology, along with some implementation details, can be found in the associated research papers (Corrado et al., 2021), (Harmon et al., 2021).

Examples of *hp*-Refinement:

The following example shows a few of the *hp*-refinement methods on the `Mesh` structure and how they might be used in practice:

```
use fem_2d::prelude::*;
let mut mesh = Mesh::from_file("./some_mesh.json").unwrap();

// isotopically h-refine all elements
mesh.global_h_refinment(HRef::t()).unwrap();

// anisotropically p-refine all elements
    //(+2 in the u-direction, +4 in the v-direction)
mesh.global_p_refinement(PRef::from(2, 4)).unwrap();

// anisotropically h-refine all elements connected
    // to some target_node in the v-direction
let target_node_id = 5;

mesh.h_refine_with_filter(|elem| {
    if elem.nodes.contains(&target_node_id) {
        Some(HRef::v())
    } else {
        None
    }
}).unwrap();

// positively p-refine all elements on the border of the mesh,
    // and negatively p-refine all other elements
mesh.p_refine_with_filter(|elem| {
    if elem.edges.iter().any(|edge_id| {
        mesh.edges[*edge_id].is_boundary()
```

```
        }) {
            Some(PRef::from(1, 1))
        } else {
            Some(Pref::from(-1, -1))
        }
    }).unwrap();
```

Example of Problem Definition

FEM_2D also features a straightforward path to extending its functionality into other problem domains. The following example shows how a simplified formulation of the Maxwell Eigenvalue Problem maps to the corresponding code in the library. This is intended provide a general depiction of how one might translate a mathematical problem into an FEM_2D implementation.

The Maxwell eigenvalue problem has the following Continuous-Galerkin formulation for an arbitrary Domain terminated with Dirichlet boundary conditions, (constraining the solution to TE modes only):

Find a solution:

$$\mathsf{U} = \{\mathbf{u}, \lambda\} \in B_0 \times \mathbb{R} \tag{1}$$

which satisfies:

$$b(\mathbf{u}, \phi) = \lambda a(\mathbf{u}, \phi) \quad \forall \phi \in B \tag{2}$$

$$\text{where: } \begin{cases} B \subset H_0(\text{curl}; \Omega) \\ a(\mathbf{u}, \phi) = \langle \nabla_t \times \mathbf{u}, \nabla_t \times \phi \rangle \\ b(\mathbf{u}, \phi) = \langle \mathbf{u}, \phi \rangle \end{cases} \tag{3}$$

The system matrices for this example are populated using FEM_2D's galerkin_sample_gep method, invoked with three generic arguments:

```
// Generate a domain (Ω) from a Mesh which has been refined as necessary
let domain = Domain::from(mesh);

// Formulate a generalized Eigenvalue problem
let gep = domain.galerkin_sample_gep::<KOLShapeFn, CurlCurl, L2Inner>(None);
```

The generic arguments correspond to the three lines of Equation 3

1. The curl-conforming basis B, which must implement the ShapeFn Trait. In this case KOLShapeFn is used.
2. The integral associated with the Stiffness Matrix (A). This argument must implement the Integral trait. In this case, CurlCurl is used.
3. The integral associated with the Mass Matrix (B). This argument also must implement the Integral trait. In this case, L2Inner is used.

The eigenvalue problem can be solved for some target eigenvalue with the following code:

```
// Dense solution (not recommended for large problems)
let solution = nalgebra_solve_gep(gep, target_eigenvalue).unwrap();

// OR: Sparse solution (requires external Slepc solver)
let solution = slepc_solve_gep(gep, target_eigenvalue).unwrap();
```

In summary, `FEM_2D` is useful for solving any generalized eigenvalue problem using FEM, simply by creating a custom implementation of the `Integral` trait. A custom Basis set can also be used by implementing the `ShapeFn` trait. Both Traits are described in detail in the crates.io documentation.

Additionally, the library has important ancillary functionality, such as solution plotting, and mesh plotting.

References

Arndt, D., Bangerth, W., Blais, B., Fehling, M., Gassmöller, R., Heister, T., Heltai, L., Köcher, U., Kronbichler, M., Maier, M., Munch, P., Pelteret, J.-P., Proell, S., Simon, K., Turcksin, B., Wells, D., & Zhang, J. (2021). The `deal.II` library, version 9.3. *Journal of Numerical Mathematics*, *29*(3), 171–186. https://doi.org/10.1515/jnma-2021-0081

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., Kong, F., … Zhang, J. (2021a). *PETSc/TAO users manual* (ANL-21/39 - Revision 3.16). Argonne National Laboratory.

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E. M., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., Kong, F., … Zhang, J. (2021b). *PETSc Web page*. https://petsc.org/. https://petsc.org/

Balay, S., Gropp, W. D., McInnes, L. C., & Smith, B. F. (1997). Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, & H. P. Langtangen (Eds.), *Modern software tools in scientific computing* (pp. 163–202). Birkhäuser Press. https://doi.org/10.1007/978-1-4612-1986-6_8

Corrado, J., Harmon, J., & Notaros, B. (2021). *A refinement-by-superposition approach to fully anisotropic hp-refinement for improved efficiency in CEM*. https://doi.org/10.36227/techrxiv.16695163.v1

Harmon, J., Corrado, J., & Notaros, B. (2021). *A refinement-by-superposition hp-method for h(curl)- and h(div)-conforming discretizations*. https://doi.org/10.36227/techrxiv.14807895.v1

Hernandez, V., Roman, J. E., & Vidal, V. (2005). SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, *31*(3), 351–362. https://doi.org/10.1145/1089014.1089019

Nalgebra: Linear algebra library for the rust programming language. (2021). In *GitHub repository*. GitHub. https://github.com/dimforge/nalgebra