FEMgine Developers' Guide

Davide Papapicco

July 14, 2023

FEMgine is an open-source project entirely written in C++ which provides the capability of discretising and solving space-time Partial Differential Equations (PDEs) using the Finite Element Method (FEM). The library is shipped as a standalone, cross-platform software package under the GNU General Public License. At this stage (date reported below the title for reference) this library is still in early pre-alpha status and development is expected to be slow but steady. As such I created this present document as a short, informal and easy-to-read guide for developers wanting to contribute to the development of the codebase.

What is the main goal and scope of this project? To create an open-source, simplified and user-friendly tool for computational scientists that are not familiar with inner workings of numerical analysis and need a quick software to simulate and validate their models using the FEM. The project was born as part of an investigation in Computational Electromagnetics (CEM) carried among the research lines of Prof. Guido Lombardi at Politecnico di Torino. To that end FEMgine has the ultimate goal of implementing Nedelec curl and div-conforming finite elements for the arbitrary-order approximation of singular vector fields.

Why should I care/contribute? The philosophy at the core of FEMgine shall also be its strength, that is simplicity and fast learning for non computationally-savvy scientists thereby providing a bridge between high-performance scientific computing and casual users of software of numerical mathematics.

Is another FEM open-source library really necessary? Absolutely not. In fact we direct the reader to large, well-established collaborative projects s.a. deal.II [1], FEniCSx [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], FreeFEM [14], Hermes [15], MFEM [16], MOOSE [17], Elmer [18], FEMPAR [19] and Gridap [20] to name but only the more famous in the international community. As mentioned before however, FEMgine puts a larger emphasis on the ease of use on the user side, by which I mean an academic of non-professional programmer that only has a minimal understanding of numerical simulations and/or computer programming.

Is there any convention about this guide? Yes there is; this guide uses a conventional colour code for highlighting and distinguishing parts of the software from the rest of the text. Specifically these boxes either indicate a file, directory or third-party software program (not necessarily integrated or used by FEMgine) while these boxes refer to FEMgine's own classes, attributes, methods etc... Finally these boxes are used to show terminal commands or specific code snippets/syntax.

Contents

1 Library structure

Upon downloading the root folder of the software presents the following directories:

- src: contains the core classes and algorithms for the implementation of the FEM;
- utilities: series of tools for support of the functioning and interface of the library;
- tutorials: test cases showing the usage of the software for different applications.

1.1 Third-party dependencies

FEMgine is standalone and open-source and thus it relies on very few third-party libraries in order to accomplish its goals, namely:

- Eigen: for high-performance matrix and linear algebra data-structures and algorithms;
- OpenGL: for post-processing and results visualisation purposes.

1.2 Building the library

The standard usage of this package is to create applications that are linked to the static library libfemgine; this is build from source using CMake to ensure portability across multiple OSs and CPU architectures.

2 Code guide

The software is written in object-oriented paradigm and each of the classes that have been implemented accomplish a unique and specific goal within the library. The general implementation philosophy throughout the library's source code is that class declaration and definition are kept separate with the former always written in an header file (.h extension) carrying the same name of the class that it declares, while the latter is always present in a source file (.cpp extension) providing all the definitions of the class' methods.

2.1 Fundamental classes

The library is composed of several classes that are all derived from two base/parent classes, namely the Entity and Mesh classes. A third fundamental class is the EntityList class although it is not inherited by any other derived class in the library.

2.1.1 The Entity class

This class is the abstraction of every structure in the codebase that is indexed within a set of similar assets. In FEM much of the heavier workload at development stage goes into the proper and efficient formatting of the data-structures and local-to-global index mapping; as such there is a need to address a diverse set of items ranging from the geometrical framework as in the Point, Line or the Element class itself to entities more to the context of numerical discretisation of PDEs s.a. DOF.

2.1.2 The EntityList class

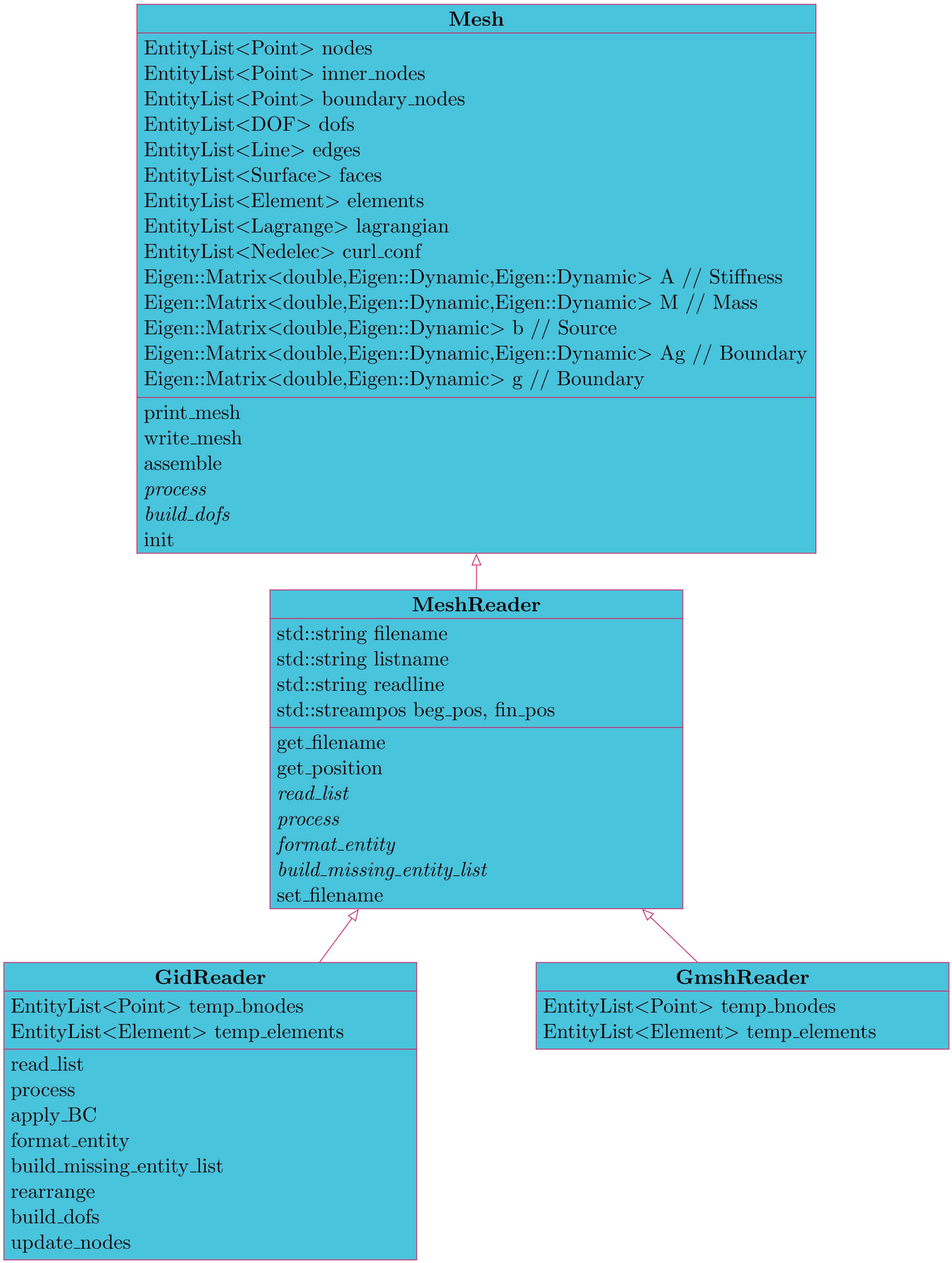
In order to efficiently store, access and operate onto the objects of the various classes derived from Entity a structure similar to the list in Python has been implemented. This class is particularly useful to keep track of the local-to-global index mapping of the items stored in it (as described above). Its fundamental (private) attribute is a std::vector< Entity > container making this a highly templatised class. It features an accessor method get_list() making it compatible with range-based loops in C++.

2.1.3 The Mesh class

As in many other scientific suites, the mesh is one of the fundamental pieces of information of the model while also being one of the most crucial, large and easy to misconceive class in OOP. This class contains a vast amount of information through its private and protected attributes while at the same time it must store such critical information in a way that can be accessed by its child classes and transformed/processed according to the user's specific needs. Of course the vast majority of its attributes are EntityList of items s.a. the various finite elements with their degrees of freedom and geometrical information as well as linear algebra operators for the assembly and the solution of the discrete linear system of equations.

2.2 Inheritance diagrams

Below there are UML-like class diagrams for the three fundamental classes.


Mesh
EntityList<Point> nodes
EntityList<Point> inner_nodes
EntityList<Point> boundary_nodes
EntityList<DOF> dofs
EntityList<Line> edges
EntityList<Surface> faces
EntityList<Element> elements
EntityList<Lagrange> lagrangian
EntityList<Nedelec> curl_conf
Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> A // Stiffness
Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> M // Mass
Eigen::Matrix<double,Eigen::Dynamic> b // Source
Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> Ag // Boundary
Eigen::Matrix<double,Eigen::Dynamic> g // Boundary
print_mesh
write_mesh
assemble
process
build_dofs
init
MeshReader
std::string filename
std::string listname
std::string readline
std::streampos beg_pos, fin_pos
get_filename
get_position
read_list
process
format_entity
build_missing_entity_list
set_filename
GidReader
EntityList<Point> temp_bnodes
EntityList<Element> temp_elements
read_list
process
apply_BC
format_entity
build_missing_entity_list
rearrange
build_dofs
update_nodes
GmshReader
EntityList<Point> temp_bnodes
EntityList<Element> temp_elements

3 Meshing

At the present moment FEMgine does not feature its own meshing utility and instead it relies on ad-hoc external software to export the data-structure and appropriately interface with.

3.1 GiD

The GiD interface is implemented through the GidReader derived class. This class is responsible for a lot of heavy-lifting in the pre-processing of the raw mesh input file as it extracts and formats all the existing information provided by Gid, namely the discrete nodes Coordinates, the vertices of the Elements and the labels and indices of the Boundary Nodes. **ATTENTION:** : in GiD's ouput raw mesh .txt file these last entities are encoded under the target Mesh Entities; the user must rename the beginning and end target as specified. It also initialises all the relevant Entity and EntityList with appropriate information and, where necessary, it computes the missing ones s.a. the list of Line and Surface delimiting the elemental domain (see the build_missing_entity_list() and build_dofs() private methods).

3.2 gmsh

The gmsh interface has not yet been implemented.

4 Pre-processing

As discussed above, FEMgine relies on external meshing software to provide the initial raw mesh data. This necessarily entails that manipulation and extraction of further information from the raw data has to be performed for a functioning application. Each mesh interface class (e.g. GidReader) implements different type of pre-processing of such data, as different outputs will be provided by the various mesh utilities.

4.1 Numbering conventions

One fundamental aspect for FEMgine is to adopt a unique, efficient and consistent notation for numbering the various Entity in the mesh. For the local numbering of the elements' vertices, edges and faces we refer to the doc/numbering_conventions subdirectory containing illustrations of the adopted convention, heavily inspired by [21] (especially for the Nedelec curl-conforming finite elements [22]).

4.2 Boundary conditions

At the moment FEMgine only supports the imposition of Dirichlet boundary conditions. The boundary function $g_D(\mathbf{x})$, $\forall \Gamma \in \partial\Omega$ is defined by the user and passed through the MeshReader interface to the library. Each edge Γ of the polygonal boundary $\partial\Omega$ is identified in the meshing facility (e.g. GiD) by a unique numerical label. The user thus associates (in FEMgine) the desired boundary function to a particular label and the apply_BC method assigns the computed value to every node in the mesh labelled appropriately.

5 Finite elements

Although the Element class derives from the Entity class, it acts as base class for the various types of finite elements to be implemented (according to Ciarlet's definition [23]). The fundamental attributes are of course EntityList encoding the geometrical information of the element however it also stores the order of approximation. **ATTENTION:** : it must be noted that the DOF list is not a member attribute as this class does not initialise a specific type of finite element. These EntityList is instead initialised by the derived classes associated to the different types of finite element (any finite element has its own type of degree of freedom). Another fundamental member attribute of this parent class is the Shape associated to the geometry of the finite element. The Shape class stores all the information relevant to the mapping between the physical element to the reference element (thus involving its Jacobian) s.a. orientation of the faces for 3D elements, the area and volume etc...

5.1 The Lagrange element

In the Lagrangian finite elements the degrees of freedom are DOF the values that the unknown numerical solution at specified points within the elemental domain (either on its boundary or within them). This class has features a parametric copy-constructor

Lagrange(Element &copy_element, EntityList<Point> arg_nodes, std::string* arg_family, int* arg_order)

that automatically initialises the correct EntityList<DOF> based on the parent object geometrical attributes and the passed-by-reference user information regarding the polynomial order of the approximation subspace.

5.2 The Nedelec element

This type of finite element is yet to be implemented.

6 Discrete linear system

The discretised linear system assembled by FEMgine has the form $\mathbf{A}\mathbf{u}_h = \mathbf{b} - \mathbf{A}_g\mathbf{g}_D$ where $\mathbf{A}$ is the stiffness matrix associated to the discretisation of the differential operator/bilinear form in the weak formulation, $\mathbf{b}$ is the source vector collecting the values of the forcing (known) function $f(\mathbf{x})$ and $\mathbf{A}_g, \mathbf{g}_D$ are the boundary matrix and vector (respectively) collecting the information at the boundary for the model. The RHS is encoded in FEMgine by the (algebric, element-wise) sum of the source vector and the product $\mathbf{A}_g\mathbf{g}_D$.

6.1 Assembly

The discretisation of the bilinear form is carried by the assemble member function of the Mesh class as it is independent on the interface with meshing utility.

6.2 Solution

The solution of the linear system is performed using Eigen built-in solvers and depending on the structure and denseness of the RHS.

7 Post-processing

The post-processing of the numerical results is not yet implemented however it shall provide error-estimation capabilities (both a-priori and a-posteriori), mesh and subspace local refinement and results visualisation.

7.1 Visualisation

The graphical visualisation of the numerical solution and its (discretised) mesh domain is yet to be implemented however it shall be performed in OpenGL.

References

[1] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The deal.II finite element library: Design, features, and insights, Computers & Mathematics with Applications 81 (2021) 407–422, development and Application of Open-source Software for Problems with Numerical PDEs. doi:10.1016/j.camwa.2020.02.022.

[2] M. W. Scroggs, J. S. Dokken, C. N. Richardson, G. N. Wells, Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes, ACM Transactions on Mathematical Software 48 (2) (2022) 18:1–18:23. doi:10.1145/3524456.

[3] M. W. Scroggs, I. A. Baratta, C. N. Richardson, G. N. Wells, Basix: a runtime finite element basis evaluation library, Journal of Open Source Software 7 (73) (2022) 3982. doi:10.21105/joss.03982.

[4] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, G. N. Wells, The FEniCS project version 1.5, Archive of Numerical Software 3 (2015). doi:10.11588/ans.2015.100.20553.

[5] A. Logg, K. Mardal, G. N. Wells, et al., Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012. doi:10.1007/978-3-642-23099-8.

[6] A. Logg, G. N. Wells, DOLFIN: automated finite element computing, ACM Transactions on Mathematical Software 37 (2010). doi:10.1145/1731022.1731030.

[7] A. Logg, G. N. Wells, J. Hake, DOLFIN: a C++/Python finite element library, in: A. Logg, K. Mardal, G. N. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method, Vol. 84 of Lecture Notes in Computational Science and Engineering, Springer, 2012, Ch. 10.

[8] R. C. Kirby, A. Logg, A compiler for variational forms, ACM Transactions on Mathematical Software 32 (2006). doi:10.1145/1163641.1163644.

[9] A. Logg, K. B. Ølgaard, M. E. Rognes, G. N. Wells, FFC: the FEniCS form compiler, in: A. Logg, K. Mardal, G. N. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method, Vol. 84 of Lecture Notes in Computational Science and Engineering, Springer, 2012, Ch. 11.

[10] K. B. Ølgaard, G. N. Wells, Optimisations for quadrature representations of finite element tensors through automated code generation, ACM Transactions on Mathematical Software 37 (2010). doi:10.1145/1644001.1644009.

[11] M. S. Alnaes, A. Logg, K. B. Ølgaard, M. E. Rognes, G. N. Wells, Unified form language: A domain-specific language for weak formulations of partial differential equations, ACM Transactions on Mathematical Software 40 (2014). doi:10.1145/2566630.

[12] R. C. Kirby, Algorithm 839: FIAT, a new paradigm for computing finite element basis functions, ACM Transactions on Mathematical Software 30 (2004) 502–516. doi:10.1145/1039813.1039820.

[13] R. C. Kirby, FIAT: numerical construction of finite element basis functions, in: A. Logg, K. Mardal, G. N. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method, Vol. 84 of Lecture Notes in Computational Science and Engineering, Springer, 2012, Ch. 13.

[14] F. Hecht, New development in FreeFem++, Journal of Numerical Mathematics 20 (3-4) (2012) 251–265.
URL `https://freefem.org/`

[15] P. Solin, L. Korous, Adaptive higher-order finite element methods for transient PDE problems based on embedded higher-order implicit Runge–Kutta methods, Journal of Computational Physics 231 (4) (2012) 1635–1649. `doi:10.1016/j.jcp.2011.10.023`.

[16] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, S. Zampini, MFEM: A modular finite element methods library, Computers & Mathematics with Applications 81 (2021) 42–74. `doi:10.1016/j.camwa.2020.06.009`.

[17] A. D. Lindsay, D. R. Gaston, C. J. Permann, J. M. Miller, D. Andrš, A. E. Slaughter, F. Kong, J. Hansel, R. W. Carlsen, C. Icenhour, L. Harbour, G. L. Giudicelli, R. H. Stogner, P. German, J. Badger, S. Biswas, L. Chapuis, C. Green, J. Hales, T. Hu, W. Jiang, Y. S. Jung, C. Matthews, Y. Miao, A. Novak, J. W. Peterson, Z. M. Prince, A. Rovinelli, S. Schunert, D. Schwen, B. W. Spencer, S. Veeraraghavan, A. Recuero, D. Yushu, Y. Wang, A. Wilkins, C. Wong, 2.0 - MOOSE: Enabling massively parallel multiphysics simulation, SoftwareX 20 (2022) 101202. `doi:10.1016/j.softx.2022.101202`.

[18] T. Gustafsson, P. Råback, J. Videman, Mortaring for linear elasticity using mixed and stabilized finite elements, Computer Methods in Applied Mechanics and Engineering 404 (2023) 115796. `doi:10.1016/j.cma.2022.115796`.

[19] S. Badia, A. Martin, J. Principe, FEMPAR: An object-oriented parallel finite element framework, Archives of Computational Methods in Engineering 25 (2018) 195–271. `doi:10.1007/s11831-017-9244-1`.

[20] F. Verdugo, S. Badia, The software design of Gridap: A finite element package based on the Julia JIT compiler, Computer Physics Communications 276 (2022) 108341. `doi:10.1016/j.cpc.2022.108341`.

[21] R. Graglia, D. Wilton, A. Peterson, Higher order interpolatory vector bases for computational electromagnetics, IEEE Transactions on Antennas and Propagation 45 (3) (1997) 329–342. `doi:10.1109/8.558649`.

[22] N. J.C., A new family of mixed finite elements in $\mathbb{R}^3$, Numerische Mathematik 50 (1986) 57–81. `doi:10.1007/BF01389668`.

[23] P. G. Ciarlet, The Finite Element Method for Elliptic Problems, Society for Industrial and Applied Mathematics, 2002. `doi:10.1137/1.9780898719208`.