Lehikoinen, A.; Davidsson, T.; Arkkio, A.; Belahcen, A.

A High-Performance Open-Source Finite Element Analysis Library for Magnetics in MATLAB

Published in:
Proceedings of the 2018 23rd International Conference on Electrical Machines, ICEM 2018

DOI:
10.1109/ICELMACH.2018.8507235

Published: 24/10/2018

Document Version
Peer reviewed version

Please cite the original version:
Lehikoinen, A., Davidsson, T., Arkkio, A., & Belahcen, A. (2018). A High-Performance Open-Source Finite Element Analysis Library for Magnetics in MATLAB. In *Proceedings of the 2018 23rd International Conference on Electrical Machines, ICEM 2018* (pp. 486-492). [8507235] IEEE.
https://doi.org/10.1109/ICELMACH.2018.8507235

A High-Performance Open-Source Finite Element Analysis Library for Magnetics in MATLAB

A. Lehikoinen, T. Davidsson, A. Arkkio and A. Belahcen

***Abstract*—Two-dimensional finite element analysis is a widely used tool in the design, analysis, and optimization of electrical machines and magnetic components. Although the core components of the method are relatively mature, numerous special applications are still being developed. This paper introduces an open-source library for finite element analysis in Matlab, tailored especially for electrical machines. The library is highly portable, and has a good performance. Furthermore, it provides all the basic functionality typically needed for magnetic analysis. As such, it can be used as a starting point for more advanced work.**

***Index Terms*—Finite element analysis, public domain software, eddy currents, motors.**

I. Introduction

Finite element analysis (FEA) has become a widely-used tool in the design and optimization of electrical machines and magnetic components alike. Indeed, numerous implementations exist, both of commercial and open-source nature. However, despite the relative maturity of general-purpose two-dimensional (2D) FEA, special applications are still seeing substantial development. Examples include magneto-mechanical analysis [1], efficient winding loss computation [2], and modelling of manufacture defects [3], to name but a few.

Several open-source FEA software and libraries exist. One of the most-commonly used is FEMM [4], which recently had a Matlab interface called `xfemm` published [5]. However, FEMM does not directly support modelling motion or non-sinusoidal time-dependency – a rather significant limitation for electrical machine design. Another option is `getDP` [6], an FEA software with its own scripting language. However, `getDP` is very general-purpose, rather than being focussed on magnetics.

Thus, this paper introduces a fully open-source FEA library for the Matlab environment, called SMEKlib (from the Finnish word SähköMEKaniikka; electromechanics). SMEKlib is tailored especially for analysing rotating electrical machines. The purpose of the library is to speed up further FEA development, a goal achieved by two main means. Firstly, the use of a high-level interpreted programming language eliminates the time spent on compilation. Arguably, it can also make the actual coding task easier, although this aspect is naturally highly person-dependent. Secondly, the library provides numerous basic building blocks needed for the analysis. This can help eliminate the need for "re-inventing the wheel", i.e. re-implementing widely-known functionality required for more advanced work. The library is freely available for download on its GitHub repository [7]. The performance of the library has been observed to be comparable to several closed-source compiled software. Thus, the term *high-performance* is best understood in the context of Matlab programs, rather than indicative of true HPC (high-performance computing) capabilities.

This paper is organized as follows. First, the overall philosophy of the library is discussed, along with some of its most important features. Next, several use-examples are given, highlighting some of the functionality. Finally, some performance and licensing aspects are briefly discussed.

The research leading to these results has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement n. 339380.

All authors are with the Dept. of Electrical Engineering and Automation, Aalto University, Finland. (email: antti.lehikoinen@aalto.fi)

II. Features and Implementation

This section provides an overview of some of the features of the SMEKlib library.

A. Philosophy

True to its name, SMEKlib is indeed a *library*. This means that significantly more attention has been given to developing the features and basic tools, rather than a user interface binding end-users to a specific workflow. As a consequence, the library is highly flexible. For example, the same matrix assembly functions can be used to analyse widely different problems, such as thermal and electrostatics.

The library aims to be as *portable* as possible. Specifically, it has no external dependencies apart from the optional `gmsh` interface discussed shortly. Furthermore, as it is implemented completely in Matlab, no installation is required apart from adding the source code to the Matlab search path.

B. Octave Compatibility

SMEKlib is largely compatible with the open-source Octave. However, the computation times have been routinely observed to be 2–4 times larger in Octave compared to Matlab. This can be attributed to the absence of a just-in-time (JIT) compiler in the current stable Octave release [8].

C. Partial List of Features

The most important features of the library are listed below. Some of the them are briefly explained immediately following the list, whereas others are covered in their respective subsections afterwards.

- Matrix and vector assembly functions for typical problems
- Typical boundary conditions
- Static, time-harmonic, and time-stepping analysis
- Linear and non-linear problems
- Circuit connections with current and voltage supply
- Arbitrary winding configurations
- Rotor movement
- Mesh generation
- Plotting and post-processing

Indeed, the library contains the necessary functionality for solving 2D magnetics problems based on the well-known Aui-formulation

$$\begin{bmatrix} \mathbf{S}+\mathbf{M}\frac{\mathrm{d}}{\mathrm{d}t} & \mathbf{C}_\mathrm{J} & \mathbf{0} \\ \mathbf{C}_\mathrm{E} & -\mathbf{I} & \mathbf{RL} \\ \mathbf{0} & \mathbf{L}^\mathrm{T} & \mathbf{Z} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{u} \\ \mathbf{i} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{u}_\mathrm{s} \end{bmatrix}. \tag{1}$$

Here, $\mathbf{S}$ and $\mathbf{M}$ denote what are commonly (also in magnetics) called the mass and stiffness matrices, and the unknowns consist of the discretized vector potential $\mathbf{a}$, the voltages $\mathbf{u}$ over the solid conductors, and the loop currents $\mathbf{i}$. [9], [10]

The current source and back-emf matrices have the entries

$$[\mathbf{C}_\mathrm{J}]_{ij} = -\frac{1}{l_\mathrm{e}} \int\limits_{\Omega_j} \sigma \varphi_i \mathrm{d}S \tag{2}$$

$$[\mathbf{C}_\mathrm{E}]_{ij} = R_i \int\limits_{\Omega_i} \sigma \varphi_j \mathrm{d}S, \tag{3}$$

where l_e is length of the problem domain in the z-direction. Furthermore, Ω_j is the domain of the conductor i, and R_i and σ denote its DC-resistance and conductivity. The DC-resistances also appear on the diagonal of $\mathbf{R}$, and $\mathbf{Z}$ contains the end-winding impedances. Finally, φ_i is the nodal shape function associated with the node i.

In nonlinear problems, (1) is first linearized with the Newton-Raphson method. With typical problem sizes, the resulting system of linear equations can then be solved with the Matlab backslash operator, utilizing the high-performance UMFPACK sparse library [11]. For uncharacteristically large problems, any of the built-in iterative solvers can be utilized.

By default, rotor movement is modelled by the moving band method [12]. Additionally, development versions for e.g. the air-gap element [13] and harmonic sliding interface [14] are available from the authors upon request, if needed for e.g. harmonic balance analysis. Additionally, a method based on polynomial interpolation [15] is an upcoming feature, inspired by the new sliding band functionality of FEMM [16].

D. Circuit Connections

In (1), the matrix $\mathbf{L}$ is the so-called loop matrix, establishing the relationship between the loop currents $\mathbf{i}$ and the total currents flowing in individual conductors. Thus, it has the entries

$$[\mathbf{L}]_{ij} = \begin{cases} +1 & \text{loop } j \text{ traverses conductor } i \text{ forwards} \\ -1 & \text{loop } j \text{ traverses conductor } i \text{ backwards} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Now, it can be seen that arbitrary winding connections – including e.g. multi-phase machines – can be analysed by simply defining a suitable loop matrix. Functions for assembling $\mathbf{L}$ for typical distributed, chorded 3-phase windings with one or two layers are included in the library.

An important special case are stranded windings. Commonly, the individual strands are not modelled as solid conductors, but an equivalent current density is used for the entire coil. In this case, only $\mathbf{a}$ and $\mathbf{i}$ will be used as unknowns. However, the relationship between them is still described by matrices similar to (2). This case will be covered later in Section III-B.

E. Vectorization and Mesh Representation

SMEKlib uses the following convention for defining and storing finite element meshes. The nodal coordinates are stored in a $2 \times N_p$ array `p`, with the first and second row corresponding to the x- and y-coordinates of the N_p nodes, respectively. Likewise, the element definitions are stored in the $3 \times N_\mathrm{e}$ array containing the indices of the nodes cornering each of the N_e elements. In use, `p` and `t` are both wrapped inside either a struct or a mesh object.

As Matlab is an interpreted languages, SMEKlib functions are largely vectorized to obtain the best performance. For example, it is well-known that the gradients of the shape functions inside the element k can be computed with the help of the reference shape function $\hat{\varphi}$, and the mapping F_k from the reference to the global element, as

$$\nabla \varphi = \mathrm{F}_k^{-\mathrm{T}} \hat{\nabla} \hat{\varphi}. \tag{5}$$

For first-order elements, the affine part of each F_k is a 2×2 matrix

$$\mathrm{F}_k = \begin{bmatrix} x_j - x_i & x_k - x_i \\ y_j - y_i & y_k - y_i \end{bmatrix}, \tag{6}$$

assuming the element k is defined by the nodes (i, j, k).

To vectorize this operation, the column-major presentation of F is first computed for all elements at once by the Matlab command

```
F = [p(:, t(2,:)) -  p(:, t(1,:));
     p(:, t(3,:)) - p(:, t(1,:))];
```

By utilizing the analytical formula for the inverse transpose of a 2×2 matrix

$$\mathbf{A}^{-T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-T} = \frac{1}{\det \mathbf{A}} = \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}, \tag{7}$$

all the gradients can be obtained in vectorized form by first computing

```
[F(4,:)*dPhiref(1) - F(2,:)*dPhiref(2,:);
F(1,:)*dPhiref(2,:) - F(3,:)*dPhiref(1)];
```

and then dividing each column of the resulting $2 \times N_e$ array by the determinant, using Matlab's `bsxfun`. This approach has been inspired by [17].

F. Boundary Conditions

Proper inclusion of boundary conditions is essential for a successful analysis. In magnetics problems, the most common conditions are probably homogeneous Neumann and Dirichlet conditions, and periodic boundary conditions. Of these, the Neumann condition is included naturally, while the latter ones must be enforced. SMEKlib largely adopts the following approach.

Indeed, both Dirichlet and (anti)periodic boundary conditions can be expressed as

$$\mathbf{a} = \mathbf{P}_\mathrm{a}\mathbf{a}_\mathrm{free}. \tag{8}$$

Here, $\mathbf{a}_\mathrm{free}$ contains the free potentials, whereas $\mathbf{P}_\mathrm{a}$ describes the mapping from free to all potentials. For example, homogeneous Dirichlet boundary nodes translate to all-zero rows in $\mathbf{P}_\mathrm{a}$. Likewise, an anti-periodic condition corresponds to a permuted $-\mathbf{I}$ block inside $\mathbf{P}_\mathrm{a}$, assuming that the nodes on the two boundaries can be mapped 1-on-1. If they cannot, a more general interpolation or mortar block has to be assembled.

With this approach, the prototypical problem (1) is transformed to

$$\mathbf{P}^\mathrm{T}\mathbf{Q}\mathbf{P}\begin{bmatrix}\mathbf{a}_\mathrm{free}\\ \mathbf{u}\\ \mathbf{i}\end{bmatrix} = \mathbf{P}^\mathrm{T}\begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{u}_\mathrm{s}\end{bmatrix}, \tag{9}$$

where $\mathbf{Q}$ denotes the left-hand-side 3×3 block matrix of (1), and

$$\mathbf{P} = \begin{bmatrix}\mathbf{P}_\mathrm{a} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}. \tag{10}$$

Now, it could be argued that (8) could be replaced by a faster indexing operation, at least in simple problems. This can indeed be true, but this would again complicate the software development process, and present an additional source of bugs. Furthermore, profiling the execution of the code will reveal that the $\mathbf{P}^\mathrm{T}\mathbf{Q}\mathbf{P}$ multiplication presents a very minor share of the total computation time, so its usage is well justified.

G. Sparse Matrix Assembly

SMEKlib adopts a fairly typical way for assembling sparse matrices. First, the non-zero entries are computed, and stored as non-ordered (row, column, value)-triplets in basic Matlab arrays. The arrays are wrapped either inside a struct, or a `MatrixConstructor` object. This way, entries can be added gradually without costly re-ordering operations which would be needed for modifying the final Matlab sparse matrix type. Once all the entries have been computed, an actual Matlab sparse matrix is assembled with the command `sparse`.

H. Mesh Generation

Being a library, SMEKlib can utilize any external mesh generator, as long as the resulting points and elements can be imported to Matlab. Indeed, the built-in geometry functionality is at the moment very limited. A built-in parametric coarse mesh generator *is* being developed as a student project, and will be briefly demonstrated later in Section III-C. General geometries, though, have to be defined externally for now.

Additionally, SMEKlib contains a simplistic interface to the popular meshing software `gmsh` [18], called `gwrap`. Included are functions for writing a `gmsh`-style geometry description file for polygonal shapes, calling the mesher, and then importing the resulting mesh to Matlab.

I. Simulation Classes

Finally, SMEKlib has several classes to shorten and simplify the workflow for analysing rotating electrical machines. The `MachineMesh` class simplifies specifying boundary conditions and setting up the moving band mesh for typical rotating machines, as well as wrapping all mesh data inside a single object. Likewise, the `MachineSimulation` class enables running harmonic and time-stepping analysis by a single method call. These will be briefly illustrated in Section III-C.

III. Usage

This section provides several use-examples of SMEKlib. The complete source code can be found in the GitHub repository [7], under `Examples`. Thus, only the most relevant parts are highlighted here.

A. TEAM Workshop

The TEAM workshop problem 30a [19] was analysed as an example problem, both with SMEKlib and the commercial software Comsol. The full example script can be found online, under `Examples/TEAM Workshop`. The folder also contains geometry data, and reference data from Comsol.

It must be noted that the induced electric field in the rotor was *not* computed from the Lorentz force, as this functionality is not currently implemented in SMEKlib. Instead, the commonly-used approach of setting the rotor frequency equal to slip times synchronous frequency was used. Thus, the results don't exactly match those given in the problem description [19].

The example script uses a mesh generated in `gmsh`, utilizing the `gwrap` functionality. The following lines demonstrate how the mesh is generated based on the geometry file `Problem30a.geo`, and then loaded to Matlab. Besides the node and element arrays `p` and `t`, also a container listing the elements belonging to each named surface is returned.

```
geo_path = [pwd '\Problem30a.geo']; %
    geometry definition file
gw = gwrap(gmsh_path); %creating gwrap
    object
```

```
gw.mesh(geo_path); %meshing geometry
[p, t, Surfaces] = gw.loadMesh(geo_path);
    %loading mesh
```

Somewhat later in the script, the following lines demonstrate a step-wise sparse matrix assembly, for fixing a uniform current density for the six conductors in the problem.

```
JF_struct = [];
for k = 1:6
    JF_struct = assemble_vector('', '
        nodal', 1, k, statorConductors{k
        }, msh, JF_struct);
end
JC = sparseFinalize(JF_struct, Np, 6);
```

The first two arguments in the function call specify that the shape functions are evaluated as such (i.e. no gradients are taken), and that nodal shape functions are used.

Since this particular problem only has homogeneous Dirichlet boundaries, the approach in Section II-F is *not* used (although it could be). Instead, the vector potential solution at each slip is obtained by simply extracting the part of the linear problem related to the non-boundary nodes `nfree`, by

```
Q = S + 1i*2*pi*fs*slip*M;
As(n_free, kw) = Q(n_free, n_free) \ FL(
    n_free);
```

Finally, Figs. 1 and 2 show the torques and rotor losses of the motor. Shown are the values obtained with SMEKlib, and by analysing the same problem in Comsol. As can be seen, a very good agreement has been obtained.

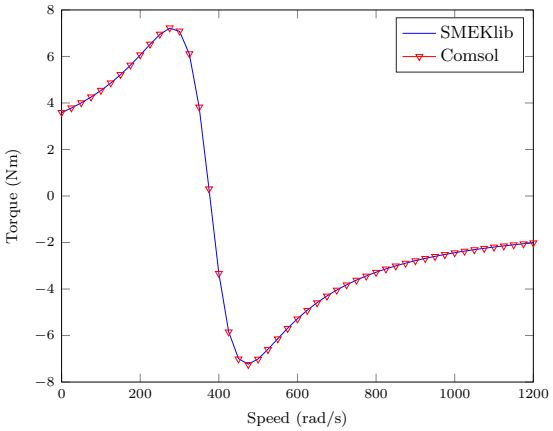

Fig. 1. Torques of the induction motor at different slips. Shown are values from SMEKlib and Comsol.

B. 37 kW Induction Motor

A somewhat more complex problem can be found under `Examples/Induction Motor (37 kW)`. A quarter of a 37 kW induction motor is analysed both in the frequency and time domains. The nonlinearity of iron is taken into account, and the voltage equations for the stator winding

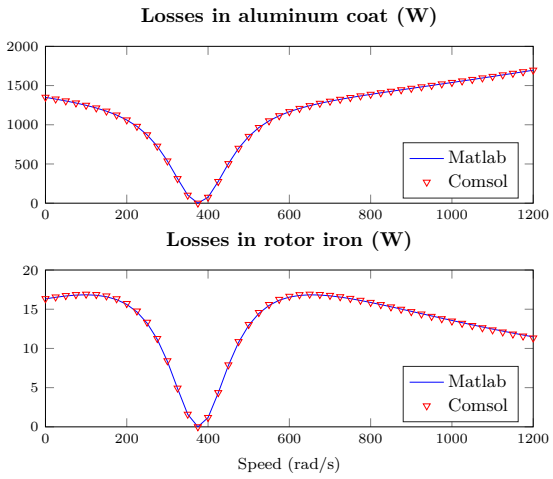

Fig. 2. Losses of the induction motor at different slips. Shown are values from SMEKlib and Comsol.

and the rotor cage are included. As a simplification, the stator winding is modelled as infinitely stranded, i.e. no eddy currents are included outside the rotor cage. The geometry of the machine can be found in Fig. 3.

The problem is set up in the `setup` script. This particular problem has an anti-periodic boundary condition $\mathbf{a}_{\text{slave}} = -\mathbf{a}_{\text{master}}$ between the symmetry sector boundaries. The boundary condition matrix $\mathbf{P}_{\text{a}}$ for enforcing this condition, as well as the Dirichlet boundary on the stator yoke, is assembled in the following lines

```
P_data = {[np_slave; np_master; msh.
    periodicityCoeff*ones(1, numel(
    np_master))], ...
     [n_dir; zeros(2, numel(n_dir))]};
P = assemble_TotalMasterSlaveMatrix(Np,
    P_data, []);
```

Here, `np_slave`, `np_master`, and `n_dir` contain the indices of the slave, master, and Dirichlet nodes, respectively. The relationships $\mathbf{a}_{\text{slave}} = -\mathbf{a}_{\text{master}}$ and $\mathbf{a}_{\text{Dir}} = \mathbf{0}$ are specified in the cell array `P_data`, after which the boundary matrix is assembled. The changing of the variable indices due to the elimination of the slave variables is handled automatically by the function.

Time-harmonic and time-stepping analysis is then performed in the scripts `timeHarmonicSimulation` and `timeSteppingSimulation` respectively. In the harmonic case, the problem (1) is split into its real and imaginary parts, to be able to solve the nonlinear problem. Furthermore, the rotor loop currents are eliminated, as this somewhat improves the condition number of the system.

In both cases, the Jacobian matrices are assembled with function calls of type

```
[J, res] = assemble_Jacobian(nu_fun, PT*
    Xfree_T, [], msh);
```

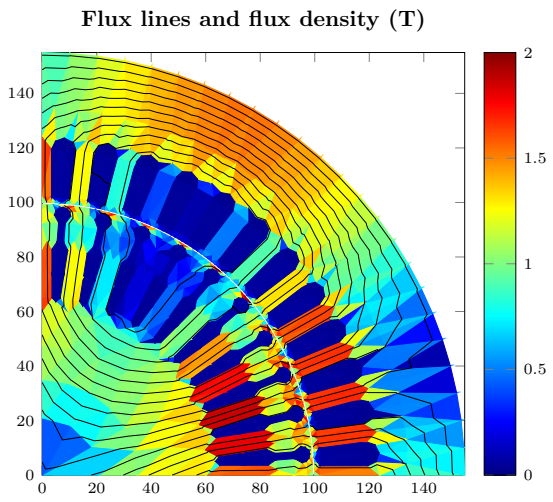

Fig. 3. Cross-section of the analysed machine, with flux lined and flux density from time-harmonic analysis. Axis units in millimetres.

Here, `nu_fun` is a handle to a function for evaluating the reluctivity of the problem materials, while the second and final argument are the vector of all unknowns, and the mesh struct/object respectively. The empty array as the third argument is a legacy artifact, and may be removed from future releases.

C. Class-Based Functionality

Finally, two examples demonstrate the simulation classes introduced in Section II-I. Firstly, the script `setup_classed` under the same `Induction Motor (37 kW)` folder contains a re-implementation of the earlier induction motor example, now with SMEKlib classes. As well as being class-based, this example utilizes Crank-Nicolson time-stepping, and contains numerous other minor improvements and modifications.

The following lines construct a `MachineMesh` object, and then set up the boundary information based on the machine dimensions `dims` supplied as an argument.

```
mshc = MachineMesh(p, t, matel);
mshc.setSymmetrySectors(4, dims);
mshc.setMachineBoundaryNodes(dims);
```

The `MachineMesh` object is then used to initialize a `MachineSimulation` object `sim`, after which time-harmonic and time-stepping analysis can be run as simply as by calling

```
sim.run_harmonic(pars); %harmonic
    analysis
sim.init(pars);
sim.run_timestepping(pars); %CN-stepping
```

The `init` method is used to compute the initial conditions for the time-stepping, as the results from harmonic analysis are typically inconsistent. Finally, `pars` is a `SimulationParameters` object, with a rather obvious purpose.

Fig. 4 then shows the steady-state phase currents obtained from the time-stepping analysis. Shown are the values obtained from SMEKlib, and a well-established in-house software FCSMEK [20]. There are obviously some minor differences, but these can be attributed to e.g. the absence of end-ring inductances in the SMEKlib model.

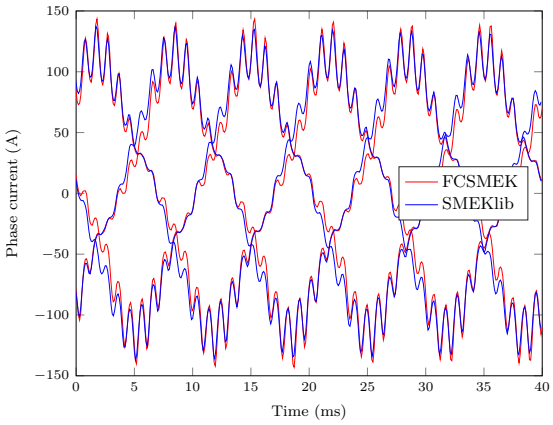

Fig. 4. Steady-state phase currents obtained from SMEKlib and an in-house software.

Finally, the script `setup_classed_meshed` introduces some under-development coarse mesh generation functionality of SMEKlib. The generated mesh can be found in Fig. 5, but otherwise the behaviour is similar to the previous example.

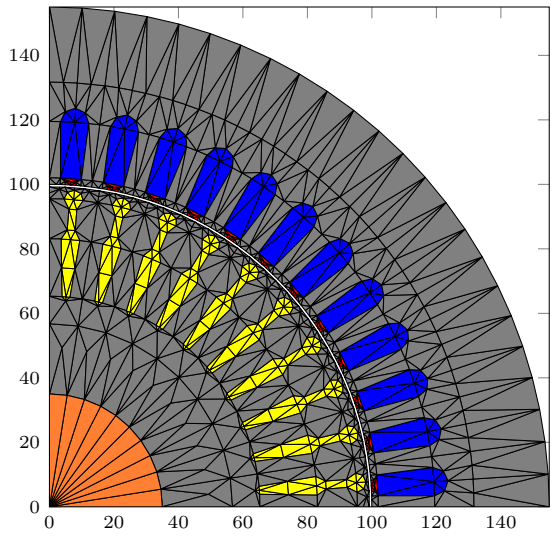

Fig. 5. Mesh for the 37 kW induction motor, generated with SMEKlib. Axis units in millimetres.

IV. Performance and Verification

As SMEKlib is largely vectorized, it can leverage the low-level linear algebra libraries used by Matlab. Indeed, it is

routinely observed to be on par with the FORTRAN-based in-house FCSMEK software, completing the 200 non-linear time-steps in Section III-C in 4.8 seconds. Furthermore, in the TEAM workshop problem of Section III-A, SMEKlib performed the actual solution step at 1.2 seconds, compared to the roughly 7 seconds taken by Comsol. Finally, SMEKlib was able to beat Infolytica's MagNet by approximately 30 % on an eddy-current problem, with the same mesh used in both software.

Naturally, a good agreement between the different software was observed in each case. However, detailed convergence analysis is beyond the scope of this article, and has been extensively covered in prior literature. Nevertheless, a logical next development step would be improving the coarse mesh generation to improve the approximation properties, as some steep flux gradients are evident to the naked eye in Fig. 3.

Admittedly, comparison with other open-source software is at the moment somewhat lacking. Nevertheless, the TEAM Workshop problem 30a *was* simulated both in Matlab and by FEMM, in the locked-rotor case. The number of unknowns was roughly 10 000 in both cases. Matlab assembled the matrices and solved the problem in 0.22 seconds, while FEMM took roughly the same time.

V. Licensing

Unlike e.g. `FEMM` or `getDP`, SMEKlib is licensed under the permissive MIT license. This means that the library can be freely modified and redistributed, even as a part of closed-source or commercial software, as long as the copyright is acknowledged.

VI. Conclusion

An open-source finite element library SMEKlib is introduced for Matlab. The library is tailored especially for modelling electrical machines, and can be used with no external dependencies. It can perform non-linear time-stepping analysis with motion and circuit connections, and visualize the results. The library is largely vectorized, and thus features good performance without any compiled components.

SMEKlib is intended to speed up the development of advanced finite element models, by providing the basic building blocks in a high-level programming language. For the same reason, it can also be used for teaching purposes.

VII. Acknowledgment

The research leading to these results has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement n. 339380.

The authors gratefully acknowledge the contributions of J. Velasco and N. Chiodetto, for their contributions on preparing the model for the TEAM problem, and analysing the library performance.

References

[1] P. Rasilo, U. Aydin, D. Singh, F. Martin, R. Kouhia, A. Belahcen, and A. Arkkio, "Multiaxial magneto-mechanical modelling of electrical machines with hysteresis," in *8th IET International Conference on Power Electronics, Machines and Drives (PEMD 2016)*, April 2016, pp. 1–6.

[2] L. Lehti, J. Kernen, S. Suuriniemi, and L. Kettunen, "Coil winding losses: Decomposition strategy," *IEEE Trans. Magn.*, vol. 52, no. 1, pp. 1–6, Jan 2016.

[3] R. Sundaria, A. Lehikoinen, A. Hannukainen, and A. Arkkio, "Higher-order finite element modeling of material degradation due to cutting," in *2017 IEEE International Electric Machines and Drives Conference (IEMDC)*, May 2017, pp. 1–6.

[4] D. Meeker, "Finite element method magnetics version 4.2 user's manual," February 2009, http://www.femm.info/wiki/HomePage.

[5] R. Crozier and M. Mueller, "A new matlab and octave interface to a popular magnetics finite element code," in *2016 XXII International Conference on Electrical Machines (ICEM)*, Sept 2016, pp. 1251–1256.

[6] P. Dular and C. Geuzaine, "GetDP reference manual: the documentation for GetDP, a general environment for the treatment of discrete problems," http://getdp.info.

[7] A. Lehikoinen and T. Davidsson, "SMEKlib - 2D-FEA library for electrical machines in Matlab," https://github.com/AnttiLehikoinen/SMEKlib.

[8] "SMEKlib - Matlab 2D-FEM library for electrical machines," https://wiki.octave.org/Classdef, accessed Jan 16 2018.

[9] I. A. Tsukerman, A. Konrad, and J. D. Lavers, "A method for circuit connections in time-dependent eddy current problems," *IEEE Trans. Magn.*, vol. 28, no. 2, pp. 1299–1302, Mar 1992.

[10] M. J. Islam, H. V. Khang, A. K. Repo, and A. Arkkio, "Eddy-current loss and temperature rise in the form-wound stator winding of an inverter-fed cage induction motor," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3413–3416, Aug 2010.

[11] T. A. Davis, "Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software*, vol. 30, no. 2, pp. 196–199, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/992200.992206

[12] B. Davat, Z. Ren, and M. Lajoie-Mazenc, "The movement in field modeling," *IEEE Trans. Magn.*, vol. 21, no. 6, pp. 2296–2298, Nov 1985.

[13] A. Abdel-Razek, J. Coulomb, M. Feliachi, and J. Sabonnadiere, "Conception of an air-gap element for the dynamic analysis of the electromagnetic field in electric machines," *IEEE Trans. Magn.*, vol. 18, no. 2, pp. 655–659, Mar 1982.

[14] H. D. Gersem and T. Weiland, "Harmonic weighting functions at the sliding interface of a finite-element machine model incorporating angular displacement," *IEEE Trans. Magn.*, vol. 40, no. 2, pp. 545–548, March 2004.

[15] X. Shi, Y. L. Menach, J. P. Ducreux, and F. Piriou, "Comparison between the mortar element method and the polynomial interpolation method to model movement in the finite element method," *IEEE Trans. Magn.*, vol. 44, no. 6, pp. 1314–1317, June 2008.

[16] D. Meeker, "Sliding band motion model for electric machines," March 2018, http://www.femm.info/wiki/SlidingBand.

[17] T. Rahman and J. Valdman, "Fast MATLAB assembly of FEM matrices in 2D and 3D: Nodal elements," *Applied mathematics and computation*, vol. 219, no. 13, pp. 7151–7158, 2013.

[18] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," *International journal for numerical methods in engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[19] K. R. Davey, "Induction motor analysis-international TEAM workshop problem 30," http://www.compumag.org/jsite/team.html.

[20] A. Arkkio, "Analysis of induction motors based on the numerical solution of the magnetic field and circuit equations," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, Dec. 1987. [Online]. Available: http://lib.tkk.fi/Diss/198X/isbn951226076X/isbn951226076X.pdf

VIII. BIOGRAPHIES

Antti Lehikoinen was born in Joensuu, Finland, in 1988. He received the B.Sc. (Tech.), M.Sc. (Tech.), and D.Sc. (Tech.) degrees in electromechanics from the School of Electrical Engineering, Aalto University, Espoo, Finland, in 2012, 2013, and 2017 respectively. Since 2017, he is a post-doc in the Research Group of Electromechanics.

His current research interests include development of computationally-efficient numerical winding loss models, the stochastic properties of circulating currents in random-wound electrical machines, as well as minimization of manufacture-related additional losses. He has also the authored the open-source FEA library SMEKlib for Matlab, and collaborated actively with various start-ups and SMEs for the development of novel electromechanical devices.

Timo Davidsson was born in Porvoo, Finland, in 1995. Currently, he is pursuing his B.Sc. (Tech.) degree in the School of Electrical Engineering, Aalto University, Espoo, Finland, and working as a part-time intern in the Research Group of Electromechanics. His current research interests include parametric model development and mesh generation.

Antero Arkkio was born in Vehkalahti, Finland in 1955. He received his M.Sc. (Tech.) and D.Sc. (Tech.) degrees from Helsinki University of Technology in 1980 and 1988. Currently he is a Professor of Electrical Engineering at Aalto University. His research interests deal with modeling, design, and measurement of electrical machines.

Anouar Belahcen (M13,SM'15) was born in Morocco, in 1963. He received the B.Sc. degree in physics from the University Sidi Mohamed Ben Abdellah, Fes, Morocco, in 1988 and the M.Sc. (Tech.) and Doctor (Tech.) degrees from Helsinki University of Technology, Finland, in 1998, and 2004, respectively. From 2008 to 2013, he has been working as Adjunct Professor in the field of coupled problems and material modeling at Aalto University, Finland. Since 2011 he is Professor of electrical machines at Tallinn University of Technology, Estonia and in 2013 he became Professor of Energy and Power at Aalto University. His research interests are numerical modeling of electrical machines, especially magnetic material modeling, coupled magnetic and mechanical problems, magnetic forces, and magnetostriction.