A new MATLAB and octave interface to a popular magnetics finite element code

Conference Paper · September 2016

DOI: 10.1109/ICELMACH.2016.7732685

2 authors:

Richard Crozier
REOptimize Systems
19 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Markus Mueller
The University of Edinburgh
166 PUBLICATIONS **3,854** CITATIONS
SEE PROFILE

A New MATLAB and Octave Interface to a Popular Magnetics Finite Element Code

Richard Crozier, Markus Mueller

***Abstract*—FEMM (Finite Element Method Magnetics) is a free, open source, high quality 2D finite element modelling tool extremely popular in academia for both research and teaching, particularly in the field of electrical machine design. However, FEMM suffers from being tied to the Microsoft Windows platform, and having a slow interface to external programs. This paper presents a successful project to extract the core algorithms from FEMM, make them cross-platform and compile them into a set of libraries and command-line programs. In addition the creation of a new Matlab and Octave interface with direct access to the libraries is described with an example of its use. The libraries use only the C++ standard template libraries for maximum portability. For students and researchers, the tools can be used without knowledge of C++, but for more advanced students and those who might want to add further improvements, the code has also been substantially reorganised for clarity. The new C++ toolbox is referred to as "xfemm", and the interface "mfemm".**

I. Introduction

FINITE Element Method Magnetics (FEMM) is a popular free and open source program for defining, solving, and post processing 2D planar and axisymmetric problems of magnetostatics, eddy currents, electrostatics and heat flow via the Finite Element Method[**?**]. FEMM is used extensively in academia and education due partly to its free and open nature, but also due to its quality. For magnetics, FEMM provides magnetostatic and quasi-static frequency analyses with non-linear domains and a range of both simple and complex boundary conditions. FEMM is also extended through the use of an embedded scripting language, Lua 4.0 which can be used in many features of the analyses.

Unfortunately FEMM is primarily a Microsoft Windows only program, although it can be run on Linux via Wine[**?**]. Although it has a fully developed method for communicating with Matlab, this is based either on ActiveX, which cannot be used on non-windows systems, or a much slower and less reliable file communication method. For these reasons it was decided to develop a version of the core FEMM numerical codes based only on the C and C++ standard libraries so it can be easily ported between systems. Instead of the current GUI, the primary interface is through Matlab/Octave[1]. However, the core code could easily be reused within a cross-platform GUI.

II. Implementation

Some details of the new implementation will now be described, followed by an overview of the new architecture of the tools.

R. Crozier and M. Mueller are with the Institute for Energy Systems, The University of Edinburgh, U.K.

[1]Octave[**?**] is an interpreted programming language with the same syntax as Matlab and largely compatible with it.

A. Core Algorithms' Conversion to C++ Standard Libraries

As stated previously, the original FEMM implementations was a Microsoft Windows only program. It used Microsoft Foundation Classes (MFC) code extensively, making it difficult to compile without access to the full professional version of the Microsoft Visual Studio. This made the software difficult to port to alternative systems. To remove this restriction, all MFC specific code was converted to use the C++ standard template libraries. Some examples of the necessary conversions were:

- Extensive use of the `CArray` type to the `std::vector` type.
- Use throughout of the `CStdString` type replaced with the `std::string` type.
- Use of the MFC `stdafx` header.
- Replacement of MFC `BOOL` type with `std::bool`

Naturally, in this process, it was impossible to convert the graphical user interface code directly from the MFC code to a cross-platform format. However, in the future, an alternative user interface based on a cross-platform framework (such as the Qt framework) could be developed. Until then, the primary interface with the C++ code is through Matlab or Octave code.

B. The C++ Code Reorganisation

Two programs and a library have been created by making modifications to the original FEMM source code. These are fmesher, fsolver and fpproc respectively. fmesher meshes a domain as specified by a normal `.fem` file (the problem description format used by FEMM). In the original FEMM program, the majority of the work of meshing a domain is handled by a high quality meshing routine named Triangle[**?**], [**?**]. In FEMM the triangle program is called as an external program. With fmesher, Triangle is instead compiled and linked to directly as a library. The fsolver program loads the mesh and problem files generated by fmesher and solves the finite element problem. fsolver and fmesher are command-line only, however, in addition to being accessible as command line programs, fmesher and fsolver can also be compiled as libraries making their functionality available to other programs. This is exploited in the creation of the direct Matlab/Octave mex interface.

fpproc is a library which provides access to the post-processing code from FEMM. As FEMM is interactive, this cannot be used directly like fsolver and fmesher, unless some custom code is used around it. A small example program is provided with the distribution to demonstrate its use in this case. However, its current primary use is in the Matlab/Octave interface.

TABLE I
PROBLEM STRUCTURE CONTENTS.

Field	Contents
ProbInfo	Information on the type of the problem, e.g. axisymmetric or planar, depth of simulation, the units used etc.
Materials	Information on materials used in the problem.
BoundaryProps	Information on boundary types used in the problem.
Nodes	Information on the node locations and properties.
Segments	Information on the segments linking nodes and their properties.
ArcSegments	Information on the arc segments linking nodes and their properties.
BlockLabels	Information used to define the properties of a region enclosed by segments and arc segments.
Circuits	Information on electrical circuits used in the model.

In addition to the creation of these main programs and libraries, a small library (named libfemm) of common code has been created to simplify and consolidate the code base which originally contained much duplication. Other improvements include renaming many classes and methods to better reflect their purpose, and a substantial increase in code comments and explanation, an effort that will continue.

C. The Matlab/Octave Interface

The Matlab/Octave Interface consists of two bodies of code. Direct mex[2] interfaces to the C++ libraries discussed previously, and a set of problem creation functions implemented as pure m-code (i.e. with no compilation required). These functions can be used to define a finite element problem in the form of a Matlab/Octave structure and directly create a file in FEMM's `.fem` format without using either FEMM, or the C++ interface.

1) Problem Creation: In FEMM, the Finite element problems are defined by creating a minimum a set of nodes, linked by segments and arc segments, labels for bound regions, and defining boundaries which are assigned to segments and arc segments. In xfemm these same elements describing a problem are stored in a conventional Matlab/Octave structure with the fields shown in Table **??**. Most of these fields are each also a structure or array of structures with information on each segment, boundary type, or other element type.

2) Mex Interface Functions: The C++ programs fmesher, fsolver and fpproc can be accessed directly from Matlab/Octave functions, being linked through the use of compiled mex functions[3]. For fmesher and fsolver, the interface is conventional and the functions simply call the underlying C++ code to first generate the mesh files, and then process these mesh and problem files respectively. The mex functions for this functionality are named `mexfmesher` and `mexfsolver`, wrappers to perform the entire process with some error checking are also provided, and these are not likely to be used directly in most cases. For post-processing the results however, a similar approach would require repeatedly loading and unloading the output files, therefore a relatively new form of interface to C++ objects has been employed. This interface wraps a Matlab handle class around a C++ object in memory and carefully handles construction, destruction and passing of information between the object and Matlab. This allows the results file produced by the solver to be loaded, and the existing post-processing functions provided by FEMM to be accessed from the interface without keeping large duplicate arrays in memory. The advantage of this is that it allows the use of the extensively tested FEMM algorithms, and avoids recreating the functionality in Matlab code.

[2]mex files are shared libraries which can be invoked from within Matlab or Octave as if they were built-in function.

[3]mexfunctions are the functions provided by mex libraries

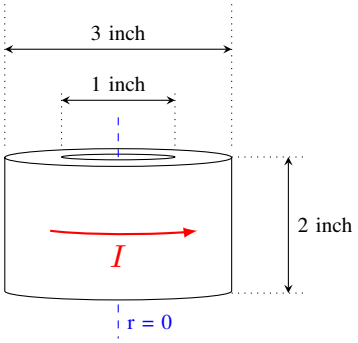

Fig. 1. Axisymmetric current-carrying coil.

III. USAGE

To demonstrate the usage of the tools, an example will now be presented of an analysis of an axisymmetric coil. This example is based on the canonical example provided as a tutorial with the original FEMM program. The problem in this case consists of an air-cored solenoid situated in free space. The arrangement is shown in Figure **??**.

The coil has an inner diameter of 1 inch; an outer diameter of 3 inches; and an axial length of 2 inches. The coil is built out of 1000 turns of 18 AWG copper wire. For the purposes of this example, we will consider the case in which a steady current of 1 Amp is flowing through the wire. In FEMM (and therefore mfemm), one models a slice of the axisymmetric problem. By convention, the r = 0 axis is understood to run vertically, and the problem domain is restricted to the region where r = 0. In this convention, positive-valued currents flow in the into-the-page direction. A suitable boundary condition for the outer boundary of the problem to approximate an infinite external region is FEMM's "asymptotic boundary condition". This condition approximates the impedance of an unbounded, open space. This boundary condition is of the form:

$$\frac{1}{\mu_0 \mu_r}\frac{\mathrm{d}\,A}{\mathrm{d}\,n} + c_0 A + c_1 = 0 \qquad (1)$$

where A is magnetic vector potential, μ_r is the relative magnetic permeability of the region adjacent to the boundary, μ_0 is the permeability of free space, and n represents the direction normal to the boundary. For the asymptotic boundary condition, the coefficients must be specified such that

$$c_0 = \frac{1}{\mu_0 \mu_r R} \tag{2}$$

$$c_1 = 0 \tag{3}$$

In this case, we will choose the region bounded by this to be a sphere with a radius of 4 inches.

3) Problem Creation Script: The following code listing shows an example of how to create the problem and solve it to generate a `.ans` file for postprocessing and viewing.

```
% Create a new FemmProblem structure for the problem.
% The first argument is either a string or scalar
% value.  Here we use `axi' to choose an axisymmetric
% simulation type. Note the length units of 'inches'.
% Other length units are available.
FemmProblem = ...
  newproblem_mfemm('axi', ...
                   'Frequency', 0, ...
                   'LengthUnits', 'inches');

% Add a wire material to the problem for use later
% a large database of materials is present, and new
% materials are easy to add
FemmProblem = ...
  addmaterials_mfemm( FemmProblem, '18 AWG' );

% Add a circuit for later use with a 1A current
FemmProblem = ...
  addcircuit_mfemm( FemmProblem, 'Coil', ...
                    'TotalAmps_re', 1 );

% Now start creating the problem geometry
outernodes = [ 0, -4;
               0,  4 ];

[FemmProblem, nodeinds, nodeids] = ...
  addnodes_mfemm( FemmProblem, ...
                  outernodes(:,1), ...
                  outernodes(:,2) );

[FemmProblem, arcind] = ...
  addarcsegments_mfemm( FemmProblem, ...
                        nodeids(1), ...
                        nodeids(2), ...
                        180, ...
                        'MaxSegDegrees', 2.5);

[FemmProblem] = ...
  addsegments_mfemm( FemmProblem, ...
                     nodeids(1), ...
                     nodeids(2) );

coilnodes = [ 0.5, -1;
              1.5, -1;
              1.5, 1;
              0.5, 1 ];

[FemmProblem, nodeinds, nodeids] = ...
  addnodes_mfemm( FemmProblem, ...
                  coilnodes(:,1), ...
                  coilnodes(:,2) );


[FemmProblem, seginds] = ...
  addsegments_mfemm( FemmProblem, nodeids(1:end), ...
                     [nodeids(end), nodeids(1:end-1)]);

% Add various labels to the problem defining regions
% properties, this is where we use the materials
% added to the problem in an earlier step
FemmProblem = ...
  addblocklabel_mfemm( FemmProblem, 0.5, 1.5, ...
                       'BlockType', 'Air', ...
                       'MaxArea', 0.1 );

% convenience functions such as addrectregion_mfemm
% (creates block with a label) ease the construction
% process
CoilBlockProps.BlockType = '18 AWG';
CoilBlockProps.InCircuit = 'Coil';
CoilBlockProps.Turns = 1000;
CoilBlockProps.MaxArea = 0.1;

FemmProblem = ...
  addrectregion_mfemm( FemmProblem, ...
                       0.5, ...
                       -1, ...
                       1, ...
                       2, ...
                       CoilBlockProps);

% Add boundary condition and assign to the arc segment by
% changing the `BoundaryMarker' to the same name as the
% new boundary.
inch = 0.0254;
R = 4;
mu_0 = 4 * pi *1e-7;
[FemmProblem, boundind, boundname] = ...
  addboundaryprop_mfemm( FemmProblem, 'ABC', 2, ...
                         'c0', 1/(mu_0*R), ...
                         'c1', 0 );

FemmProblem.ArcSegments(arcind).BoundaryMarker ...
  = boundname;
```

4) Visualise the Problem: The presented geometry is quite simple, but for more complex geometries, a method of visualising the problem during construction is desirable. Two methods of doing this are provided with the tools. The first is a simple function to open the problem in the original FEMM program, if it is present on the system. The function `openprobleminfemm_mfemm` writes the FemmProblem structure to a temporary `.fem` file and calls FEMM to open the file for viewing. It would simply be called as

```
openprobleminfemm_mfemm(FemmProblem);
```

Alternatively, a native m-code plotting tool is also provided to view the geometry with an example plot shown in Fig. **??**.

```
plotfemmproblem(FemmProblem);
```

5) Problem Solution: The problem is solved with a simple function call and data can easily be extracted at multiple points in a natural way using an object oriented syntax. For example, to retrieve the flux density at a set of points the following syntax may be used:

```
myfpproc.getpointvalues([0, 0.1, 0.2], ...
                        [0,   0,   0])
```

The flux density results (in the r and z directions) are returned in a matrix with each point's values in its own column.

```
ans =

     0.0000e+000    -7.1020e-006   -11.0640e-006
    17.9056e-003    17.9404e-003    18.0306e-003
```

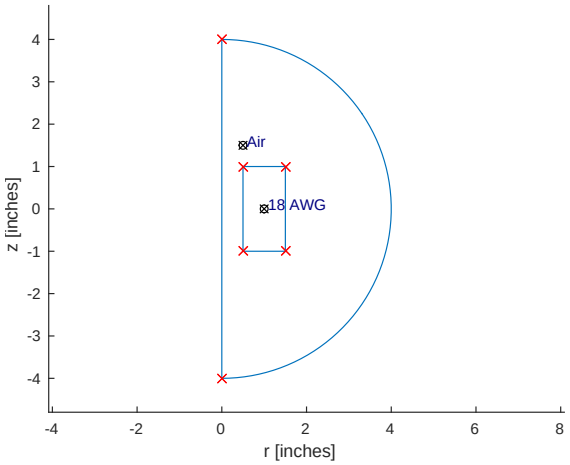

Fig. 2. Coil geometry plot in Matlab.

Integrals on regions can also easily be performed. As an example, a region can be selected and the cross-sectional area can be found with the following code.

```
myfpproc.selectblock(1, 0, true);

coilarea = myfpproc.blockintegral(5)

coilarea =

    1.2903e-003
```

6) Visualising Results: As with the `.fem` files, the `.ans` files produced by mfemm are fully compatible with FEMM and may be viewed from within the program. However, some native visualisation tools are also provided with mfemm. An example plot of the magnetic flux density created by mfemm is presented in Fig. **??**. Fig. **??** shows a filled contour plot of the vector potential.

A. More Advanced Example

The ability to more easily and natually interface with Matlab code makes it easier and therefore quicker to develop more complex simulations that exploit the rest of the Matlab computing system. For example Fig. **??** shows a geometry of a slotted radial flux permanent magnet machine developed in Matlab and fully parametrised. This has been used to determine the flux linkage waveform presented if Fig. **??** by performing multiple magnetostatic simulations with different relative rotor and stator positions. A piecewise cubic curvefit has then been performed on the waveform, allowing the derivative to easily be extracted and used to evaluate the EMF and determine interesting parameters such as the harmonic content. In this case the machine is modelled using periodic boundary conditions on segment edges and has radially magnetised magnets. FEMM has the ability to also apply a magnetisation based on a formula defined as a Lua language variable string, a feature which xfemm retains, along with all other advanced features available in FEMM.

The fitting procedure and geometry creation are not part of the interface package, but were facilitated by it's existence. A large package of electrical machine simulation tools has now been developed based on the interface, but a description of this software is beyond the scope of this paper.

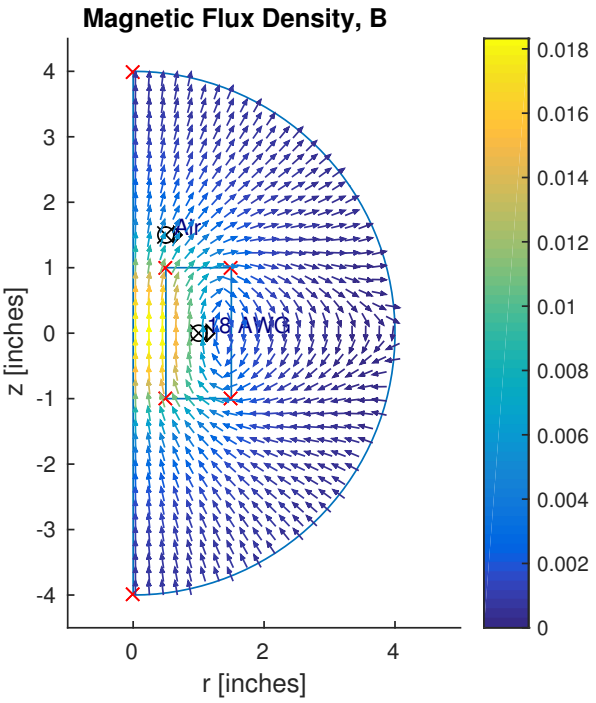

Fig. 3. Magnetic flux density vector field plot with geometry in Matlab.

B. Other Useful Features

Like FEMM, xfemm includes a thermal solver. However, as problems are constructed in a standard Matlab structure, the process of converting an existing geometry to a thermal simulation is considerably eased. Thermal and magnetic material properties coexist within the same problem description, as can thermal and magnetic boundary conditions. Fig. **??** shows a thermal simulation of the same machine geometry used in the previous example. To produce this thermal simulation required minor amendments to the magnetic simulation structure, such as the the removal of the outer regions and the addition of appropriate thermal boundary conditions on the new outer boundary. Modifications to the geometry are facilitated by a collection of functions for locating geometric elements (e.g. `findsegment_mfemm`), and by the fact that the problem is defined in an easily modifiable Matlab Structure.

IV. Performance

The performance of problem solving and meshing are similar to the original FEMM. The primary gains in performance are in the speed of problem creation and post-processing from m-code. Previously these would be performed through an ActiveX interface (on Windows only). On Linux it is necessary to use a much slower and less reliable file-based communication method. Speedups in problem creation can be 10 times or more with the new Matlab interface as each command does not have to be sent through the interface, but

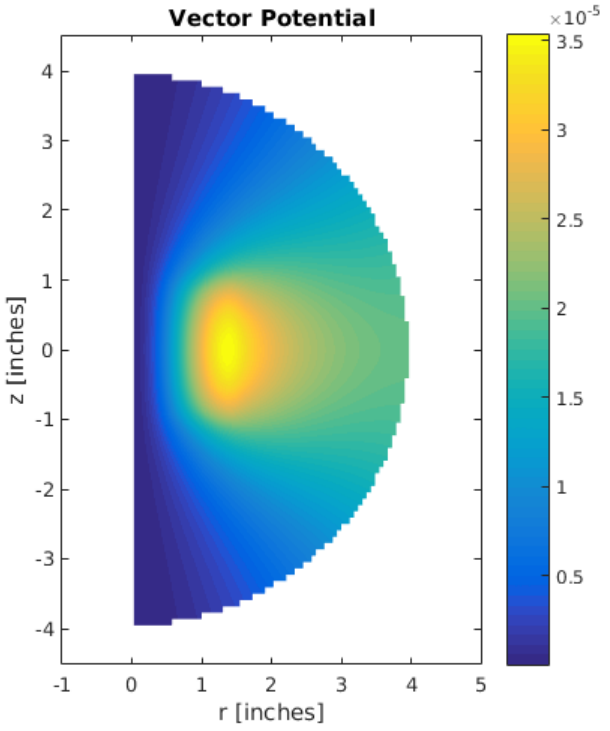

Fig. 4. Magnetic vector potential field plot in Matlab.

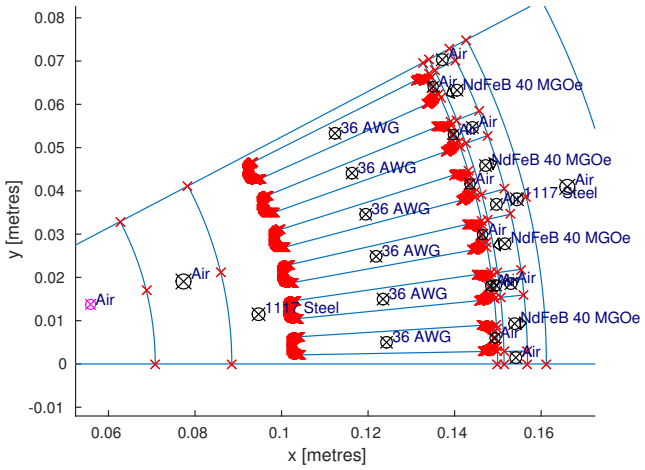

Fig. 5. Radial flux permanent magnet machine geometry plot.

this is heavily dependant on the nature of the simulation. For large batch runs of small problems, the message passing in both the problem creation and data extraction phases could dominate the total computation time when using the old Matlab interface.

V. Licencing

The licensing arrangements of xfemm are slightly complex as the program is made up of several components of slightly differing licensing terms. The C++ programs based on the original FEMM source code are licensed under the Aladdin Free

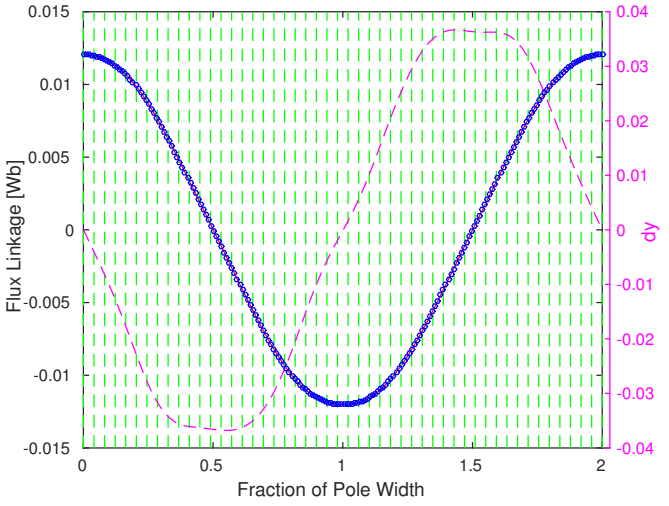

Fig. 6. Flux linkage waveform in one coil of the machine shown in Fig. **??** with 1st derivative with respect to normalised position relative to a machine pole. The vertical lines denote the 'knot' points where the piecewise cubic curves join.

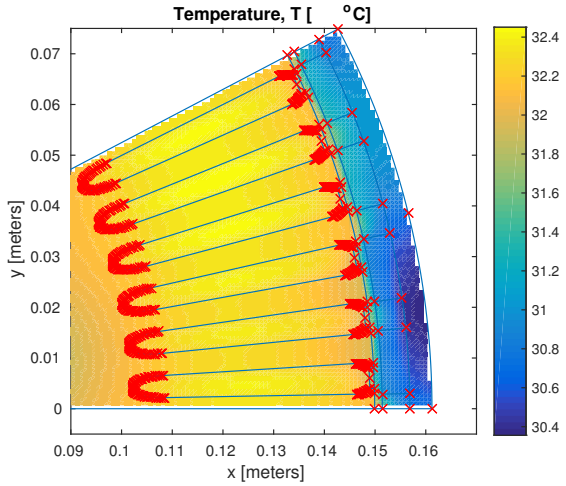

Fig. 7. Plot of thermal simulation output for the same machine as shown in Fig. **??**.

Public License, as the original FEMM source is also provided under this license. The Matlab/Octave code is provided under the Apache Version 2.0 licence which is another open source licence which allows redistribution and copying freely. Further details and the texts of these licenses are provided with the source.

VI. Conclusion

A new interface has been presented to the core algorithms of the popular finite element analysis program Finite Element Method Magnetics. With the Windows specific code replaced with only standard template library functions, the algorithms are now fully cross-platform, and may be compiled natively on other operating systems. This opens a number of possibilities for their use not previously possible. As an example of this, the described direct interface to the code from the Matlab

and Octave programming languages has been developed, that greatly increases the speed of problem creation and post-processing in comparison the ActiveX based interface supplied with the original FEMM. Furthermore, in this format, batch runs of simulations are generally more easily performed, and parallelism more easily exploited when a large number of simulations are to be performed.

By improving the clarity of the code base, it is also easier for students and researchers to gain an understanding of how the code operates, and consider modifying or extending it. It also facilitates future interfaces to other popular languages, e.g. Python, or lower-level languages such as C++ or Fortran, also allowing the code to be used efficiently as a component of a larger simulation. At the time of writing, the source code is freely distributed by the author online[**?**].

ACKNOWLEDGMENT

The authors would like to thank Dr. David Meeker for authoring the original FEMM and freely providing the source code.

Richard Crozier Richard Crozier studied Electrical Engineering at The University of Edinburgh and was later awarded a PhD from the same university on the modelling of direct-drive linear machines for wave energy applications. He then worked as a research associate developing the electromagnetic design of a prototype linear generator technology referred to as 'Snapper', a project for which the technical work was rated as excellent by the European Commission. Since then he has worked in close collaboration with industry performing knowledge transfer and assisting with the development of both wave and tidal energy technologies. Most recently he has been on secondment for two years to a tidal energy developer designing a direct-drive generator for their future drive train. Richard has an interest in the development of open-source engineering software tools and has been a developer for a range of projects including the Qucs circuit simulation software, xfemm, and an electrical machine simulation toolbox, which is under construction.

Markus Mueller Prof. Markus Mueller studied Electrical Engineering at Imperial College and awarded a PhD from the University of Cambridge on the modelling of induction motors. Subsequently he worked at Cambridge as a research assistant collaborating with Brook Crompton Motors. After 2 years with SR Drives Ltd as a Senior Development Engineer, he took up a lectureship at the University of Durham in 1997. In January 2004 he moved to the University of Edinburgh and is now the Head of Institute at The Institute for Energy Systems. His main research interests lie in the design of electrical machines for renewable energy converters. In 2006 he was awarded the Donald Julius Groen prize by the IMechE for his work on direct drive linear generators for wave energy converters. He has published just over 100 papers, and is the inventor of C-GEN for which 2 patents have been filed.