David B. Davidson
Dept. E&E Engineering
University of Stellenbosch
Stellenbosch 7600, South Africa
Tel: +27 21 808 4458;
Fax: +27 21 808 4981
E-mail: davidson@sun.ac.za

Foreword by the Editor

In the December 2000 and August 2003 columns, the development of FEM codes for electromagnetics was discussed. Around a decade on, some new tools have been developed in computational science and engineering for the application of the FEM. This month's paper addresses the use of one of these, namely the *FEniCS* package, for the FEM solution of electromagnetic problems.

As the editor is one of the authors, this submission was handled editorially by Leo Kempel, a former Associate Editor of the *Magazine*. This contribution is gratefully acknowledged by the editor. Furthermore, the authors would like to thank him for his insightful review.

Using the *FEniCS* Package for FEM Solutions in Electromagnetics

A. J. Otto[1], N. Marais[2], E. Lezar[3], and D. B. Davidson[4]

Computational **E**lectro**MAG**netics **G**roup – **CEMAGG**
Department of Electrical and Electronic Engineering
University of Stellenbosch
Western Cape, South Africa 7600
E-mail: [1]mail@braamotto.com; [2]nmarais@gmail.com; [3]mail@evanlezar.com; [4]davidson@sun.ac.za

Abstract

FEniCS is a set of software tools that allows for rapid implementation of expressions associated with finite-element analysis. The main interface for *FEniCS* is *DOLFIN*, which provides both *C++* and *Python* front ends to the software tools. The use of the *Python* front end, *PyDOLFIN*, to model various electromagnetic (EM) problems is investigated. Some elementary problems implemented in *FEniCS* include the scalar potential solution to closed- and open-boundary electrostatic problems, as well as the cutoff and dispersion analysis of a hollow rectangular waveguide. More advanced topics considered include the implementation of radiation from an infinitesimal dipole, and near-field-to-far-field transformations.

Keywords: Computational electromagnetics; DOLFIN; Finite Element Method; FEniCS; Python

1. Introduction

The *FEniCS* Project [1] is a collection of free software with an extensive list of features for automated, efficient solution of differential equations. In this paper, the use of *FEniCS* in the implementation of the Finite-Element Method (FEM) in electromagnetic (EM) applications is investigated. First, several elementary problems are addressed. These include the scalar potential solution to closed- and open-boundary electrostatic problems, as well as the cutoff and dispersion analysis of a hollow rectangular waveguide. Second, a selection of some more advanced topics is considered, including the implementation of an infinitesimal-dipole radiation problem, and near-field-to-far-field transformations. Previous papers in this column have addressed FEM code development using the development tools then available. In [2], many practical aspects of FEM programming for vector elements, using primarily *FORTRAN*, were addressed. In [3], a comprehensive overview of meshing and linear-algebra packages then available, and especially sparse-matrix routines, was presented.

2. The *FEniCS* Project

FEniCS is a set of software tools that allows for the rapid implementation of the expressions associated with the finite-element analysis of problems from a wide array of disciplines. The main interface of the software system is *DOLFIN*, which provides both a *C++* and a *Python* front end. The use of the *Python* front end (*PyDOLFIN*) in modeling electromagnetic problems is considered here. One of the strengths of *DOLFIN* is that it provides built-in support for a number of finite-element families, including the curl-conforming Nédélec (of the first and second kind [4, 5]) and Lagrange finite-element spaces, used in vector and scalar formulations, respectively [6, 7]. *FEniCS* has some built-in support for generating FEM meshes, but also supports importing meshes from other packages, such as *Gmsh* [8].

3. Solving Electrostatic Problems

To introduce the use of *FEniCS* in the implementation of the FEM in EM applications, the solution of the two-dimensional (2D) electrostatic scalar potential function is discussed. In particular, we consider the Poisson equation, given in [9] as

$$-\nabla\bullet\left(\varepsilon\nabla\bullet\varphi\right)=\rho\,, \tag{1}$$

where ε, φ, and ρ are the electric permittivity, electric scalar potential, and electric charge density, respectively. It is well known that the principle of solving boundary-value problems (BVP) using the FEM is to subdivide a continuous domain into a number of subdomains. In these subdomains, an unknown scalar potential function, φ, is represented by interpolation functions with unknown coefficients. In addition to the differential equation in Equation (1), the unknown potential, φ, is required to satisfy a number of boundary conditions [9]:

$$\varphi=\varphi_D \ \text{ on } \Gamma_{\mathrm{D}}\,, \tag{2}$$

$$-\frac{\partial\varphi}{\partial n}=g \ \text{ on } \Gamma_N\,, \tag{3}$$

$$\frac{\partial\varphi}{\partial n}+p\varphi=pq \ \text{ on } \Gamma_R\,. \tag{4}$$

Here, $\Gamma=\Gamma_D+\Gamma_N+\Gamma_R$ is the boundary enclosing the domain area, Ω, and $\frac{\partial}{\partial n}$ represents the partial derivative with respect to the boundary normal. The constants φ_D, g, p, and q are known parameters or functions describing the physical properties of the boundary, and are problem dependent. The boundary condition on Γ_D as in Equation (2) is referred to as a Dirichlet or essential boundary condition. For such a boundary condition, the value for the scalar potential is given a prescribed value (in this case, φ_D), whereas for Neumann boundary conditions as in Equation (3), the derivative of the scalar potential is prescribed. Neumann boundary conditions with $g=0$ are referred to as natural boundary conditions, and are enforced as part of the variational process. The boundary condition on Γ_R in Equation (4) is a Robin boundary condition or boundary condition of the third type, and represents a generalization of the Neumann condition.

The Poisson equation in Equation (1), along with the boundary conditions discussed, make up the boundary-value problem (BVP) that is to be solved. It can be shown that the solution of this boundary-value problem using the Finite-Element Method and a Galerkin procedure results in the following form, which was also used in [10]:

$$\frac{1}{2}\int_{\Omega}\varepsilon\nabla L_i\bullet\nabla L_j d\Omega+\int_{\Gamma_N}\varepsilon gL_j d\Gamma +\frac{1}{2}\int_{\Gamma_R}\varepsilon pL_iL_j d\Gamma-\int_{\Gamma_R}\varepsilon pqL_j d\Gamma=\int_{\Omega}\rho L_j d\Omega\,. \tag{5}$$

The functions L_i and L_j are scalar Lagrange basis functions that are used as the trial and testing functions in the Galerkin procedure, with the unknown scalar potential, φ, being written as a weighted sum of these basis functions. Note that to arrive at Equation (5), the assumption has been made that the electric permittivity, ε, is isotropic and constant in an element. The application of Equation (5) to all combinations of testing and trial functions yields the following matrix equation:

$$\left[A\right]\left\{c\right\}=\left\{b\right\}, \tag{6}$$

with the elements of the matrix $\left[A\right]$ and the column vector $\left\{b\right\}$ being given by [10]

$$(A)_{ij} = \int_{\Omega} \varepsilon \nabla L_i \bullet \nabla L_j d\Omega + \int_{\Gamma_R} \varepsilon p L_i L_j d\Gamma , \quad (7)$$

$$(b)_{ij} = \int_{\Omega} \rho L_j d\Omega - \int_{\Gamma_N} \varepsilon g L_j d\Gamma + \int_{\Gamma_R} \varepsilon p q L_j d\Omega . \quad (8)$$

The elements of the vector $\{c\}$ are the unknown coefficients of the basis functions used in the discretization of the scalar potential, φ [6]. Note that the Robin boundary condition of Equation (4) contributes to both $[A]$ and $\{b\}$.

The discussion presented here serves only to summarize the key points related to the formulation of scalar electrostatic problems using the FEM. For more information and an in-depth treatment of the subject, the reader is referred to texts such as [6, 7, 9, 11].

3.1 Example Problems

The electrostatic boundary-value problems considered in this discussion are shown in Figure 1.

The first problem is the uniform potential distribution of two parallel plates spaced a distance D apart, as shown in Figure 1a. The top plate was at a known fixed potential, V, and as such, a Dirichlet boundary condition of $\varphi = V$ was applied. On the bottom (grounded) plate, the boundary condition was $\varphi = 0$. It is important to note that this two-dimensional (2D) representation assumed infinitely large plates, and, due to the symmetry in the parallel-plate configuration, the open boundaries on the sides could be considered as homogeneous Neumann boundary conditions ($g = 0$ in Equation (3)).

The second configuration considered, shown in Figure 1b, was that of a coaxial cylinder with the center conductor at a potential V, and the outer conductor at ground potential. This configuration only had Dirichlet boundary conditions, as in Equation (2). Both the coaxial cylinder and parallel-plate configurations are both considered bounded-domain (or closed-boundary) problems. The solution of these types of problems using *FEniCS* is discussed in Section 3.2.

The third configuration was a conductor at a height, H, above a ground plane, as shown in Figure 1c. In this example, the conductor was at a known fixed potential ($\varphi = V$), while the ground plane was at zero potential ($\varphi = 0$). This is considered to be an unbounded-domain (or open-boundary) problem, as the physical solution exists in an infinite space around the wire conductor. As such, the solution method has to take this radiating property into consideration. There are two ways to approach this. The first is to simply make the solution domain very large, and apply zero-valued Dirichlet boundary conditions on all the outer boundaries [11]. If the boundaries are chosen far enough away from the region of interest – around the wire at height H – this method can yield reasonable results, but can quickly become computationally expensive. The second approach is to define a Robin boundary condition as in Equation (4), with p and q containing information about the physical properties of the boundary, and allowing for the description of the radiation through this boundary. The definition of a mixed-boundary-condition problem in *FEniCS* is discussed in Section 3.3.

3.2 Defining the Closed Boundary Electrostatic Problem in *FEniCS*

The description of the electrostatic problem in *FEniCS* started with the domain discretization. Although *FEniCS* has a built-in mesh generator as discussed in Section 4, an open-

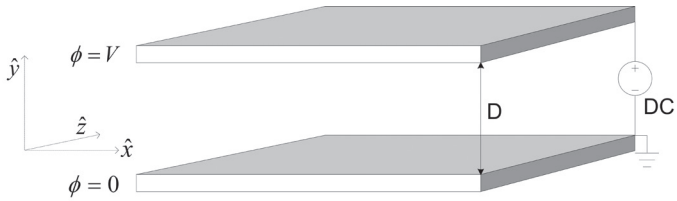

Figure 1a. The parallel plate electrostatic configuration.

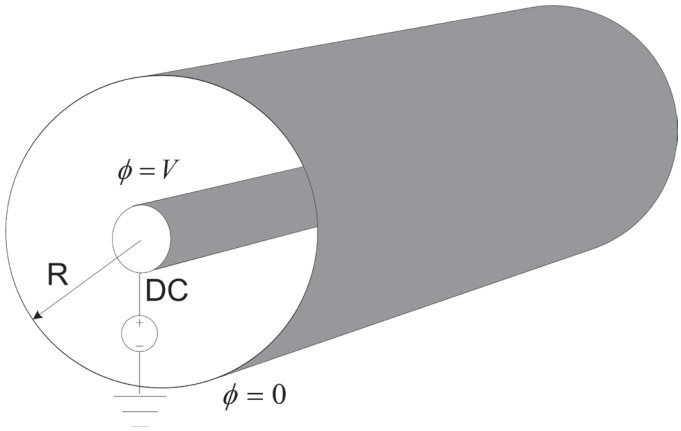

Figure 1b. The coaxial cylinder electrostatic configuration.

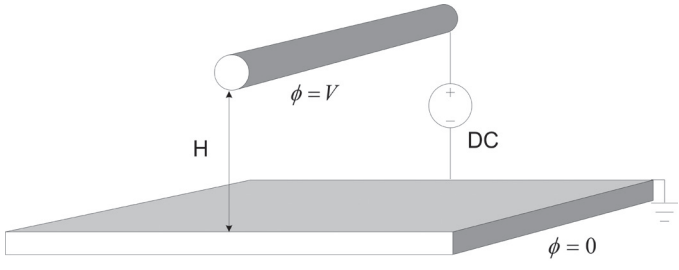

Figure 1c. The wire-plane electrostatic configuration.

source finite-element grid generator, called *Gmsh* [8], was used for the electrostatic examples. Furthermore, in *Gmsh*, markers can be assigned to nodes and edges, and allowance is made for the easy description of boundary contours such as Γ_D, Γ_N, and Γ_R. The function `dolphin-convert` converts the *Gmsh* file format to an Extensible Markup Language (XML) that *DOLFIN* can interpret. This mesh and corresponding markers can be converted for use in *DOLFIN* as shown in Figure 2. In this example, the mesh for the parallel-plate problem has been shown, but the extension to the other problems is straightforward.

The next step in the *FEniCS* computational process is to initialize the function space used in the finite-element discretization. As discussed, the Lagrange basis functions are included as part of *FEniCS* to an arbitrary order. The function space, as well as test and trial functions, were defined as shown in Figure 3.

The Dirichlet boundary conditions now have to be enforced on the nodal function space. The boundary conditions on the sources and ground were set to 1 V and 0 V, respectively, in the code shown in Figure 4. Here, the *DOLFIN* `DirichletBC` class, in conjunction with the `Constant` class, valued 0 and 1, were used. The boundary-condition classes were also initialized with the markers defined in *Gmsh*, with the ground boundary condition applied to nodes marked with value 2, and the source boundary condition to nodes marked with value 1.

For the parallel-plate and coaxial-cylinder problems, $g = 0$, $p = 0$, and $q = 0$ for the Neumann and Robin boundary conditions of Equations (3) and (4). Furthermore, if the charge density in Equation (1) is also zero – yielding the Laplace equation – then $\rho = 0$ in Equation (5), and the integrals over Γ_N and Γ_R in Equation (5) also evaluate to zero. The matrix and vector elements of Equations (7) and (8), respectively, can then be computed in *FEniCS* as shown in Figure 5. Here, `dx` represents integration over the domain Ω, and the relationships between `A_ij` and Equation (7) and `b_ij` and Equation (8) should be immediately evident.

With the boundary conditions and the matrix elements defined, it was possible to assemble the matrices using the *DOLFIN* assemble function, apply the boundary conditions, and obtain the solution for the problem as shown in Figure 6.

It is interesting to compare the above code to the *MATLAB* script in [11, Figure 10.2], which solved a similar electrostatic problem. The much higher level of coding afforded by *FEniCS* is apparent. Note that the `apply` method of a boundary condition ensures that the relevant degrees of freedom in the matrix or vector of the associated linear system are assigned the correct values. In this example, the `solve` function solved the linear system of Equation (6), with the result stored in the coefficients of the *DOLFIN* `Function phi`. Although this solution could be plotted using the built-in *FEniCS* `plot` function by calling `plot(phi)`, for these electrostatic examples, the solution was written to a disk to allow for visualization using an open-source package called *Paraview* [12], as shown in Figure 7. Figure 8 shows the results obtained in this way for the closed boundary problems of the coaxial cylinder and infinitely large parallel plate.

3.3 Defining the Mixed Dirichlet and Robin Open-Boundary Problem in *FEniCS*

The mixed Dirichlet and Robin open-boundary electrostatic problem again starts with the domain discretization. The importing of the converted mesh file, the identification of the markers, and the definition of the function space are all similar to those actions described in Section 3.2. First-order Lagrange basis functions were again used to define the trial and test functions. The Dirichlet boundary conditions were enforced on the nodal function space by setting the source conductor to a constant potential of 1 V, while the boundary conditions on the ground plane were set to a constant potential of 0 V. The source markers in *Gmsh* were defined as "1," the ground plane (boundary) markers were defined as "2," while the open-boundary markers were defined as "3."

As seen in Section 3.2, the electrostatic Laplace form of the Poisson equation has $\rho = 0$ as a constant. The open-boundary condition electrostatic problem now had a mixed Dirichlet and Robin boundary condition. The weak form of this problem in standard notation $a\left(L_i, L_j\right) = L\left(L_i\right)$ therefore yields

$$a\left(L_i, L_j\right) = \int_{\Omega} \varepsilon \nabla L_i \bullet \nabla L_j d\Omega + \int_{\Gamma_R} \varepsilon p L_i L_j d\Gamma , \qquad (9)$$

$$L\left(L_i\right) = \int_{\Omega} \rho L_j d\Omega + \int_{\Gamma_R} \varepsilon pq L_j d\Omega . \qquad (10)$$

It was discussed in [9] that if the problem has an open or unbounded domain, the domain can be truncated by defining a fictitious surface that encloses the primary domain of interest. If the fictitious surface is far enough away from the excitation, an approximate homogeneous boundary condition of the third type can be defined as given in Equation (11):

$$\frac{\partial \varphi}{\partial r} \approx \frac{1}{r \ln r} \varphi , \qquad (11)$$

where $r = \sqrt{x^2 + y^2}$. The Robin boundary conditions were only on the open boundaries that were defined by markers "3" in *Gmsh*.

The essential boundary conditions were enforced and the system could be solved in the usual way, as shown in Figure 10. Figure 11 shows the results obtained for the mixed open-boundary problems of the wire above a ground plane.

```
# import the required modules
from dolfin import *
# Load the mesh
mesh=Mesh("parallel_plate.xml")
# import the Gmsh defined markers
markers=MeshFunction('uint',mesh,'parallel_plate_facet_region.xml')
```

Figure 2. The mesh and marker definitions after converting *Gmsh* to XML format for use in *DOLFIN*.

```
# define the order of the function space
scalar_order=1
# define the scalar nodal function space (nodal_space) as a Lagrange function space
nodal_space = FunctionSpace(mesh, 'Lagrange', scalar_order)
# define the test and trial functions from the nodal_space, with L_i and L_j the Lagrange
basis functions
(L_i) = TestFunctions (nodal_space)
(L_j) = TrialFunctions (nodal_space)
```

Figure 3. The definition of the function space, and the test and trial functions.

```
# Define the Dirichlet boundary condition for ground
bc_ground = DirichletBC(nodal_space, Constant(0.0), markers, 2)
# Define the Dirichlet boundary condition for the source
bc_source = DirichletBC(nodal_space, Constant(1.0), markers, 1)
```

Figure 4. The definition of the source and ground reference Dirichlet boundary conditions.

```
# rho=0 is a constant for the electrostatic Laplace form of Poisson equation
rho = Constant(0.0)
# define left hand side of weak form equation
A_ij = inner(grad(L_i), grad(L_j))*dx
# define right hand side of weak form equation
b_ij = rho*L_j*dx
```

Figure 5. The definition of the matrix and vector elements given by Equations (7) and (8).

```
# Assemble the matrices
A = assemble ( A_ij )
b = assemble ( b_ij )
# apply the boundary conditions for ground and the source
bc_ground.apply ( A, b )
bc_source.apply ( A, b )
# setup the solution
phi = Function ( nodal_space )
# let c be the coefficients for the solution
c = phi.vector()
# solve the linear system
solve ( A, c, b )
```

Figure 6. The matrices are assembled, boundary conditions are applied, and a solution to the linear system is obtained.

```
# write solution of phi to file to open and plot in Paraview
file = File('parallel_plates.pvd')
file << phi
```

Figure 7. The solution is written to a file to open and plot in *Paraview*.

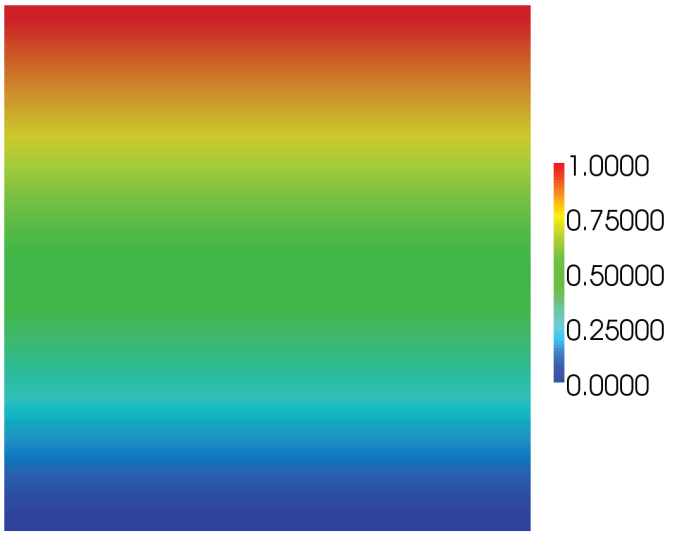

Figure 8a. The electrostatic potential distribution for the infinitely large parallel-plate configuration.

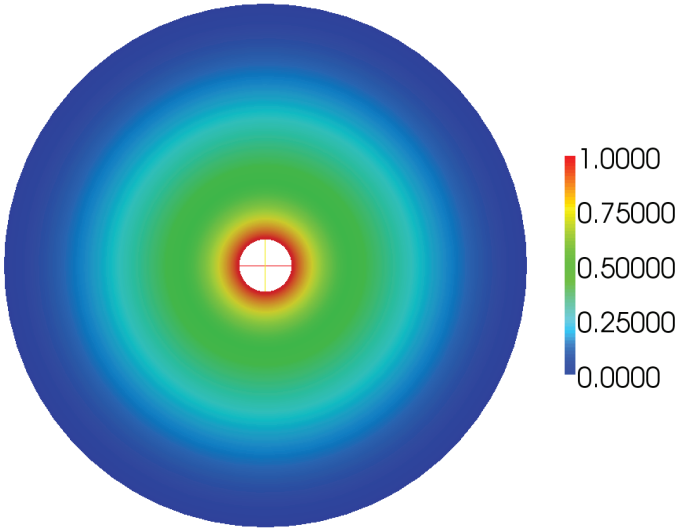

Figure 8b. The electrostatic potential distribution for the coaxial-cylinder configuration.

3.4 Analytical Compared to Numerical Results

The analytical expression for the potential distribution between parallel plates is given in Equation (12):

$$V_{\text{Parallel Plates}} = \frac{QD}{\varepsilon_0 A}, \tag{12}$$

where Q is the total charge on the top source plate in coulombs, D is the distance between the plates in meters, $\varepsilon_0 = 8.854 \times 10^{-12}$ F/m is the permittivity of free space, and A is the area of the plates in m^2. The analytical potential distribution for the parallel plates, compared to the *FEniCS* computed distribution for a constant $\hat{x}$ position and varying height ($\hat{y}$ position), is shown in Figure 12a.

The analytical expression for the potential distribution between coaxial conductors is given as

$$V_{\text{Coaxial Cylinder}} = \frac{Q}{2\pi\varepsilon_0} \ln\left(\frac{R}{r}\right), \tag{13}$$

where R is the radius of the outer conductor in meters, and r is the radius of the inner conductor in meters. The analytical potential distribution for the coaxial conductors, compared to the *FEniCS* computed distribution for a varying radial ($\hat{x}$) position away from the center conductor's surface, is shown in Figure 12b.

The analytical expression for the potential distribution in the area surrounding a wire above an infinitely large ground plane is given by

$$V_{\text{Wire-Plane}} = \frac{2H}{r}, \tag{14}$$

where H is the height of the conductor above the ground plane in meters, and r is the radius of the conductor in meters. The result of the analytical potential distribution for the wire-plane

```
# rho=0 is a constant for the electrostatic Laplace form of Poisson equation
rho = Constant(0.0)
# p is calculated as approximate boundary condition d(phi)/dr = 1/(r*ln(r))*phi
r = Constant (2.0)
p = 1/(r*ln(r))
# q is as a source which in this example is 0
q = Constant ( 0.0 )
# define left hand side of weak form equation - ds(3) refers to integration over the open boundary represented by markers 3
a = inner(grad(L_i), grad(L_j))*dx + p*L_i*L_j*ds(3)
# define right hand side of weak form equation - ds(3) refers to integration over the open boundary represented by markers 3
L = rho*L_j*dx + p*q*L_j*ds(3)
# assemble the A and b matrices on the domain to solve A[phi]=b
A = assemble(a, exterior_facet_domains=markers)
b = assemble(L, exterior_facet_domains=markers)
```

Figure 9. Defining the left- and right-hand sides of the weak-form equation, and assembling the matrices.

```
# enforce essential boundary conditions
for bc in bcs:
    bc.apply(A, b)
# define the function of unknowns to solve in nodal function space
phi = Function(nodal_space)
P = phi.vector()
# solve the system of matrices
solve ( A, P, b )
```

Figure 10. Enforcing the boundary conditions and solving the system of matrices.

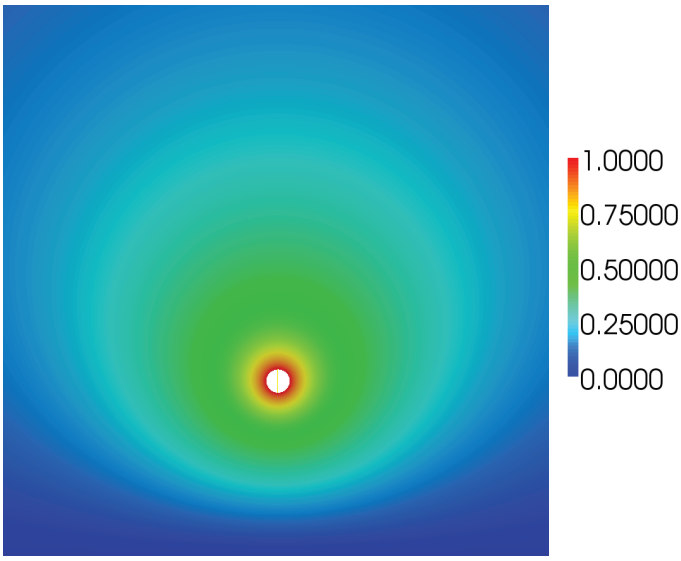

Figure 11. The electrostatic potential distribution for the wire-plane configuration.

configuration compared to the *FEniCS* computed distribution for a constant height ($\hat{y}$ position), H, and varying $\hat{x}$ position away from the conductor's surface is shown in Figure 12c. In all three examples, good agreement was found between the computed and analytical results. In Figure 12c, the analytical solution assumed an infinitely large ground plane of infinite extension in the transverse direction, whereas the FEM solution used a large, but finite, ground plane.

4. Waveguides

As an introduction to the use of *FEniCS* in the cutoff and dispersion analysis of waveguides, consider the canonical problem of a hollow rectangular waveguide, as shown in Figure 13. In the full-wave analysis of such guides, it is required to solve the vector Helmholtz equation [9], along with the relevant boundary conditions. The time-independent versions of these are given by

$$\nabla\times\frac{1}{\mu_r}\nabla\times\vec{E}-k_0^2\varepsilon_r\vec{E}=0 \text{ in } \Omega, \tag{15}$$

$$\hat{n}\times\vec{E}=0 \text{ on } \Gamma_e, \tag{16}$$

$$\hat{n}\times\nabla\times\vec{E}=0 \text{ on } \Gamma_m, \tag{17}$$

where Ω is the domain represented by the interior of the waveguide, and Γ_e and Γ_m are electric and magnetic walls, respectively. The parameters μ_r and ε_r are the relative permeability and permittivity, respectively, of the medium inside the guide, and may be position dependent. The free-space wavenumber, k_0, is related to the operating frequency, f_0, by [13]

$$k_0=\frac{2\pi f_0}{c_0}, \tag{18}$$

with c_0 being the speed of light in free space.

It can be shown that in the case of cutoff analysis, the finite-element solution results in the following matrix equation [9, 11]:

$$\begin{bmatrix}S_{tt} & 0\\ 0 & S_{zz}\end{bmatrix}\begin{Bmatrix}e_t\\ e_z\end{Bmatrix}=k_c^2\begin{bmatrix}T_{tt} & 0\\ 0 & T_{zz}\end{bmatrix}\begin{Bmatrix}e_t\\ e_z\end{Bmatrix}, \tag{19}$$

or simply

$$[S]\{e\}=k_c^2[T]\{e\}. \tag{20}$$

The matrices $[S]$ and $[T]$ are described by their submatrices, the elements of which are given as follows:

$$(S_{tt})_{ij}=\int_\Omega\frac{1}{\mu_r}\left(\nabla_t\times\vec{N}_i\right)\bullet\left(\nabla_t\times\vec{N}_j\right)d\Omega, \tag{21}$$

$$(T_{tt})_{ij}=\int_\Omega\varepsilon_r\vec{N}_i\bullet\vec{N}_j d\Omega, \tag{22}$$

$$(S_{zz})_{ij}=\int_\Omega\frac{1}{\mu_r}\left(\nabla_t L_i\right)\bullet\left(\nabla_t L_j\right)d\Omega, \tag{23}$$

$$(T_{zz})_{ij}=\int_\Omega\varepsilon_r L_i L_j d\Omega. \tag{24}$$

Matrix equations such as these are in the form of a generalized eigenvalue problem [11]. The eigenvalues and eigenvectors of the system are related to the cutoff modes – the cutoff wavenumber and basis-function coefficients, respectively – and

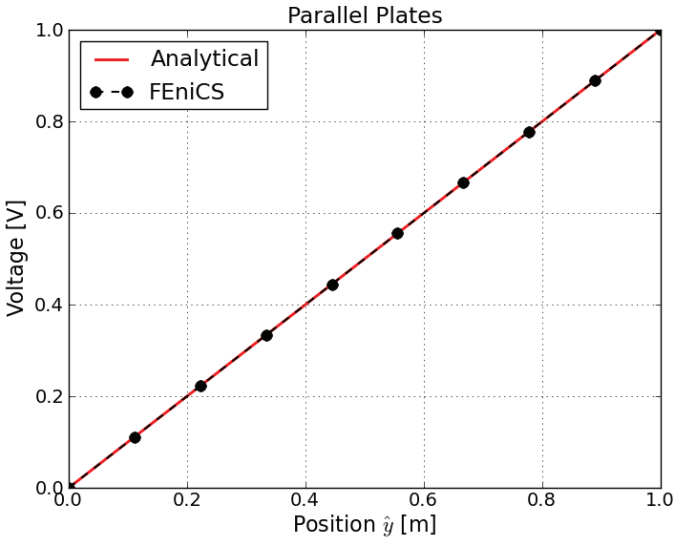

Figure 12a. The analytically computed compared to the *FEniCS* computed electrostatic potential distributions for the parallel-plate configuration.

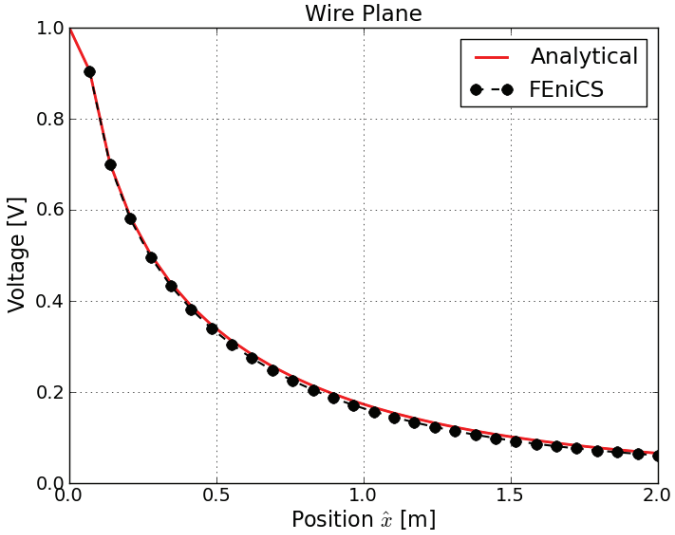

Figure 12c. The analytically computed compared to the *FEniCS* computed electrostatic potential distributions for the wire-plane configuration.

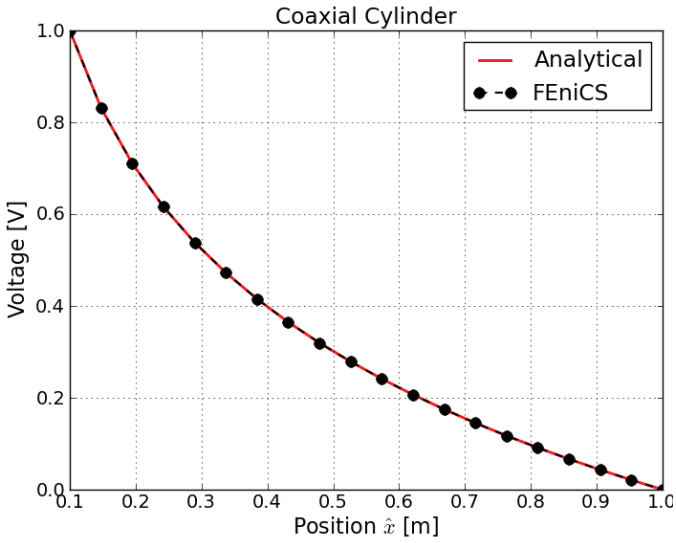

Figure 12b. The analytically computed compared to the *FEniCS* computed electrostatic potential distributions for the coaxial-cylinder configuration.

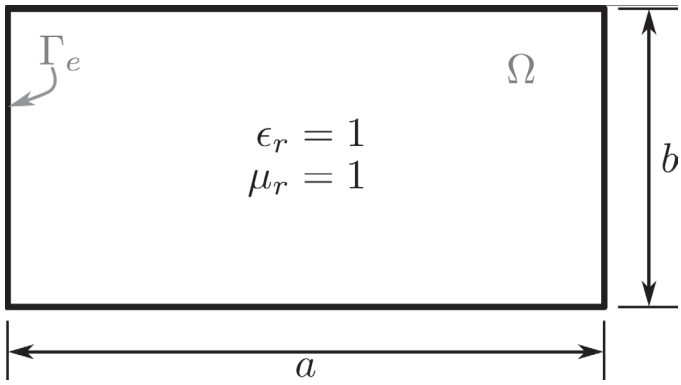

Figure 13. The cross section of a hollow rectangular waveguide, with width, a, and height, b. Also shown is the computational domain, Ω, as well as the perfectly electrically conducting boundary, Γ_e. The relative permittivity and permeability (ε_r and μ_r, respectively) inside the guide were both unity.

are the quantities of interest in this case. Since the purpose of this paper is not to revisit the finite-element formulations of any of the problems presented, Equations (21)-(24) were given without derivation (the reader is directed to texts including [9, 11, 14] for in-depth discussions on the subject).

Readers familiar with electromagnetic finite-element formulations should recognize Equations (21) to (24) as various forms of the elemental representations of the finite-element mass and stiffness matrices. It should be further noted that both the vector and scalar formulations are present, with $\vec{N}_i$ and $\vec{N}_j$ representing Nédélec curl-conforming basis functions [4, 5], and L_i and L_j representing scalar Lagrange basis functions used in [6], for example.

In Equations (19) to (22), $\int_{\Omega} \bullet d\Omega$ represents integration over the cross section of the waveguide, and ε_r and μ_r are the relative permittivity and permeability in the domain, respectively.

4.1 Cutoff Analysis

With a recap of the relevant equations completed, it is possible to start with the description of the problem in *FEniCS*, and to work our way through to the solution of Equation (20). The first step was to define the discretization, as shown in Figure 14. Here, both the mesh and function spaces were defined. For the former, the *DOLFIN* class `Rectangle` was used, while the `FunctionSpace` class was instantiated with the relevant parameters for the Nédélec (of the first kind) and Lagrange function spaces, respectively. Second-order vector basis functions and third-order nodal basis functions were chosen in this example, but *FEniCS* can handle higher orders. Note that each `FunctionSpace` also receives the mesh as a parameter. Since the mixed formulation discussed at the start of this section was employed, a combined function space was set up, and the basis functions (the `TestFunctions` and `TrialFuctions` classes) were taken from this space.

With the discretization defined, it was possible to go ahead and define the forms (loosely interpreted as the expressions for the matrix elements) associated with Equations (21) to (24), and to assemble the relevant matrices. As already mentioned, one of the most attractive aspects of *FEniCS* is the ease with which these expressions can be implemented. In the cutoff-analysis case, the matrices of Equations (23) and (24) could be constructed for an unloaded guide ($\varepsilon_R = \mu_r = 1$) as shown in Figure 15. In this listing, there should be a clear relationship between the definitions of the forms for the matrix elements – such as `s_tt_ij` – in the code and the expressions for the matrix elements already discussed. It should be noted that in *DOLFIN*, multiplication by `dx` in a form results in integration over the cells of the spatial domain when the assemble function is called. The resultant matrices `S` and `T` were *DOLFIN* Matrix objects. In the case of the hollow rectangular waveguide, the variables – `e_r` and `u_r` – representing the material parameters were simply constants equal to unity.

Before the matrices `S` and `T` could be used as part of the solution process, the boundary conditions of Equation (16) still had to be applied. For this, the `DirichletBC` class was used, with the mechanism used to specify the boundary edges shown in Figure 16. Here, `MeshFunction` was a list of unsigned integers (`uint`) associated with each of the edges in `mesh`. After initializing `boundary_markers` to zero for each edge,

```
# import the required modules
from dolfin import *
import numpy as np
# the mesh used for the rectangular hollow guides
a = 1.0
b = 0.5
# create a rectangular mesh with origin (0,0) extending to (a,b) with 8 edges along the long side and 4
elements along the short side
mesh = Rectangle ( 0, 0, a, b, 8, 4 )
# define the orders of the function spaces for vector and nodal basis functions
vector_order = 2
nodal_order = 3
# define the functions spaces
vector_space = FunctionSpace ( mesh, "Nedelec 1st kind H(curl)", vector_order )
nodal_space = FunctionSpace ( mesh, "Lagrange", nodal_order )
combined_space = vector_space * nodal_space
# define the test and trial functions from the combined space
# here N_v and N_u are Nedelec basis functions and L_v and L_u are Lagrange basis functions
(N_i, L_i) = TestFunctions ( combined_space )
(N_j, L_j) = TrialFunctions ( combined_space )
```

Figure 14. The definitions of mesh dimensions, orders for vector and nodal basis functions, function spaces, and test and trial functions from the combined space.

```
# specify the relative permittivity and permeability
e_r = 1.0
u_r = 1.0
# define the forms (matrix elements) for cutoff analysis ito the basis functions
s_tt_ij = 1.0/u_r * inner ( curl_t(N_i), curl_t(N_j) )
t_tt_ij = e_r * inner ( N_i, N_j )
s_zz_ij = 1.0/u_r * inner ( grad(L_i), grad(L_j) )
t_zz_ij = e_r * L_i * L_j
# post-multiplication by dx will result in integration over the domain of the mesh at assembly time
s_ij = ( s_tt_ij + s_zz_ij ) * dx
t_ij = ( t_tt_ij + t_zz_ij ) * dx
# assemble the system matrices. DOLFIN automatically evaluates each of the forms for all the relevant test
and trial function combinations ie. all possible values of i and j
S = assemble ( s_ij )
T = assemble ( t_ij )
```

Figure 15. The definition of the forms, or matrix elements, for the cutoff analysis of the unloaded waveguide.

```
# create a mesh function to mark the edges (dimension 1) in the mesh.
boundary_markers = MeshFunction ( 'uint', mesh, 1 )
# mark all edges as 0
boundary_markers.set_all ( 0 );
# mark the edges on the boundary as 1
DomainBoundary().mark( boundary_markers, 1 )
```

Figure 16. Creating a mesh function to mark the edges and specify the boundary edges.

```
# create the boundary condition using the combined function space, a zero Expression, and the mesh function
# for the edges. note that the last parameter (1) is used to indicate the edges where the boundary condition  must be applied.
electric_wall = DirichletBC ( combined_space, Expression ( ("0.0", "0.0", "0.0") ) , boundary_markers, 1 )
# apply the boundary condition to the assembled matrices:
electric_wall.apply ( S )
electric_wall.apply ( T )
```

Figure 17. The Dirichlet boundary conditions applied to the `S` and `T` matrices.

```
# initialise a vector of ones.
indicators = np.ones ( S.size(0) )
# get the boundary indicators to remove the rows and columns associated with the boundary DOFs.
indicators[electric_wall.get_boundary_values().keys()] = 0
# the free DOFs correspond to the elements of indicators that are equal to 1.
free_dofs = np.where(indicators == 1)[0]
# convert the dolfin matrices to numpy arrays selecting only the rows and columns associated with free DOFs
S_np = S.array()[free_dofs,:][:,free_dofs]
T_np = T.array()[free_dofs,:][:,free_dofs]
```

Figure 18. Pre-processing to remove rows and columns associated with the boundary degrees of freedom.

```
# solve the eigensystem (S_np, T_np) using scipy
from scipy.linalg import eig
k_c_squared, ev = eig ( S_np, T_np )
# sort the calculated values
sort_index = np.argsort( k_c_squared )
# skip over the non-physical (zero) eigenmodes
first_mode_idx = np.where(k_c_squared[sort_index] > 1e-8)[0][0]

print "The cutoff wavenumbers of the 4 most dominant modes are:"
print k_c_squared[sort_index][first_mode_idx:first_mode_idx+4]
```

Figure 19. The *SciPy* eigensolver was used to obtain the desired cutoff wavenumbers.

```
# TE_10 mode is the first mode
mode_idx = 0
# Post-process the coefficients to map back to the full matrix
coefficients_global = np.zeros ( S.size(0) )
coefficients_global[free_dofs] = ev[:,sort_index[first_mode_idx+mode_idx]]
# Create a Function on the combined space.
mode = Function ( combined_space )
# Assign the coefficients of the function to the calculated values.
mode.vector().set_local ( coefficients_global )
# Split the function into the parts in each of the functions spaces in combined_space
# This is done using DOLFIN's Function().split()
( TE, TM ) = mode.split()
# Plot the mode using the dolfin plotter
plot ( TE, title="TE_10 mode" )
```

Figure 20. The visualization of the TE_{10} mode.

```
# TE_11 mode is the fourth mode
mode_idx = 3
# Post-process the coefficients to map back to the full matrix
coefficients_global = np.zeros ( S.size(0) )
coefficients_global[free_dofs] = ev[:,sort_index[first_mode_idx+mode_idx]]
# Create a Function on the combined space.
mode = Function ( combined_space )
# Assign the coefficients of the function to the calculated values.
mode.vector().set_local ( coefficients_global )
# Split the function into the parts in each of the functions spaces in combined_space
# This is done using DOLFIN's Function().split()
( TE, TM ) = mode.split()
# Plot the mode using the dolfin plotter
plot ( TM, title="TM_11 mode" )
```

Figure 21. The visualization of the TM_{11} mode.

the edges that fell on the mesh boundary – indicated by the `DomainBoundary` class – were marked with the value "1."

The Dirichlet boundary condition associated with the PEC surrounding the computational domain could now be created and applied to the matrices `S` and `T` as shown in Figure 17. Since the boundary was intended to represent a PEC, all the basis functions that fell on the boundary – whether they were Nédélec functions associated with an edge, or Lagrange functions that corresponded to a node – had to be set to zero, and a *DOLFIN* `Expression` with zero value was used to do this. The final two parameters in the initialization of the `electric_wall` boundary condition were the mesh functions marking the edges of the mesh and the marked value for which the boundary condition had to be applied.

Note that the `apply` method of the `DirichletBC` class in *DOLFIN* does not remove the corresponding row and column of the matrix for zero-valued boundary conditions, but instead zeros the rows of the matrix and inserts a one on the matrix diagonal. When these matrices describe a generalized eigenproblem as was the case here, the effect is that an eigenvalue with value 1.0 is added to the system for each of the zeroed rows. In order to simplify further discussions, the matrices were preprocessed to remove the rows and columns associated with the boundary degrees of freedom, as shown in Figure 18. Although *DOLFIN* does supply an eigensolver, the discussion of its use is outside the scope of this paper. Instead, the original *DOLFIN* matrices `S` and `T` were converted to `NumPy` arrays – `S_np` and `T_np`, respectively – for further processing using the array method as shown.

The eigenproblem described by the new matrices – with only the rows and columns associated with the free degrees of freedom selected – could now be solved, and the desired cutoff wavenumbers obtained. In this case, the eigensolver included as part of `SciPy` was used to solve the problem as shown in Figure 19. Here, the finite-element problem has been solved, and the variables `k_c_squared` and `ev` fully described the quantities of interest in Equation (24). The former contained a list of the square of the cutoff wavenumber for all the modes computed, whereas the corresponding columns in `ev` were the coefficients for the free degrees of freedom that allowed for the modal-field distribution to be calculated.

Although the cutoff wavenumbers could simply be printed out, the visualization of the modal distributions needed some additional processing. Take, as examples, the TE_{10} and TM_{11} modes – modes 0 and 3, respectively – which corresponded to the first purely transverse and transverse-axial modes for the waveguide dimensions considered. The first step in the post-processing was mapping the entries of `ev` (associated with the free degrees of freedom) back to the global degrees of freedom, including those associated with the Dirichlet boundary condition (see Figure 20). After the global coefficients `coefficients_global` were obtained, these coefficients were used to define a discrete *DOLFIN* `Function` in the combined function space. Since this function could have components in both the transverse and longitudinal spaces, these had to be split, and the desired part plotted using the *DOLFIN* `plot` function.

The process for the TM_{11} mode was similar, with the only differences being that in that case, the index of the mode was 3, and the part of the combined function space in which the mode resided was different (see Figure 21).

With the computation of both the cutoff wavenumbers and mode distributions completed, it was possible to compare these to analytical results. Firstly, consider Table 1, where the computed values of the square of the cutoff wavenumbers for the first four modes are compared to the analytical values [15]. It was clear that these matched well for all four modes, with the relative error on the order of 10^{-6} for the dominant mode.

The modal distributions for the TE_{10} and TM_{11} modes are plotted in the guide cross section in Figures 22 and 23, respectively. These had the expected form, and could be compared visually to any number of texts, including [11].

5. Advanced Topics

By solving the near and far fields of an infinitesimal electrical dipole, this example demonstrates how to implement a custom source; implement a first-order Mur absorbing boundary condition (ABC); implement the near-to-far-field transform using *FEniCS* forms notation; reconstruct near-field results; and use `matplotlib` through the *PyLab* module to plot results. Since the complete code listing was fairly long, we provided the full source in the new software package *SUCEM:FEM* [16, 17], which was built on *FEniCS* and is available under an open-source license. We only highlight important parts of the code in the text. These facilities are used to solve the near- and far-fields of an infinitesimal electromagnetic dipole.

5.1 Formulation

The standard high-frequency electromagnetic FEM formulation, based on the vector Helmholtz (i.e., curl-curl) equation, and solving for the electric field was used. The reader is directed to [9, 11, 14] for detailed discussions of the subject. Since we were solving an open problem, a mesh-termination scheme was needed.

The first-order ABC [9, 11] can be seen as an impedance condition relating the tangential $\vec{E}$ and $\vec{H}$ fields on the problem boundary. By using Faraday's law ($\vec{H} = jk_0^{-1}\mu_r^{-1}Z_0^{-1}\left(\nabla\times\vec{E}\right)$), this can be expressed completely in terms of the E field:

$$\hat{n}\times\left(\nabla\times\vec{E}\right)+jk_0\hat{n}\times\left(\hat{n}\times\vec{E}\right)=0\,. \qquad (25)$$

This relation can be substituted into the standard FEM formulation, yielding (after also neglecting magnetic-current sources)

Table 1. A comparison of the computed and analytical cutoff wavenumbers squared from the TE_{10} and TM_{11} modes of a hollow rectangular guide with dimensions 1.0 m × 0.5 m.

Mode	TE_{10}	TE_{01}	TE_{20}	TM_{11}
Analytical [m^{-2}]	$1.0\pi^2$	$4.0\pi^2$	$4.0\pi^2$	$5.0\pi^2$
	9.8696	39.4784	39.4784	49.3480
Calculated [m^{-2}]	9.8696	39.4793	39.4793	49.3484
Relative error	1.45×10^{-6}	2.19×10^{-5}	2.19×10^{-5}	7.14×10^{-6}

$$\int_V \left[\left(\nabla\times\vec{T}\right)\bullet\mu_r^{-1}\bullet\left(\nabla\times\vec{E}\right)-k_0^2\vec{T}\bullet\varepsilon_r\bullet\vec{E}\right]dV = -jk_0\int_{S_0}\left(\hat{n}\times\vec{T}\right)\bullet\left(\hat{n}\times\vec{E}\right)dS - \int_V \vec{T}\bullet\left[jk_0Z_0\vec{J}_{imp}\right]dV, \quad (26)$$

where $\vec{E}$ is the electric field, $\vec{T}$ is a testing function, Z_0 is the intrinsic impedance of free space, and $\vec{J}_{imp}$ is the impressed electric current. The first term on the right-hand side is a surface integral; this integral represents the action of the ABC.

5.2 Discrete Formulation

The correct finite-element space to use is that of the curl-conforming (H(curl)) elements, also known as Nédélec elements. Typical "mass" and "stiffness" forms can be recognized in Equation (26). The ABC enters the FEM functional as a surface bilinear form involving the tangential components of the test and trial functions on the outside boundary of the problem. Since it is a bilinear form, it needs to be moved to the left-hand side of the discrete FEM equation. For use with *DOLFIN*, we needed to separate out the real and imaginary parts of the form. Assuming lossless materials, we noted that the mass and stiffness forms were purely real, while the ABC form was purely imaginary. The approach we followed was to calculate the forms as real matrices, and then combine them in *SciPy* to use complex solvers. *FEniCS* naturally deals with a surface integral differential element via `dolfin.ds`. A symbolic element-normal is also available (`n=V.cell().n`, where `V` is the *DOLFIN* `FunctionSpace` object), which can be combined with the *FEniCS* cross operator to express the bilinear form. The code representing the bilinear forms was given as shown in Figure 24.

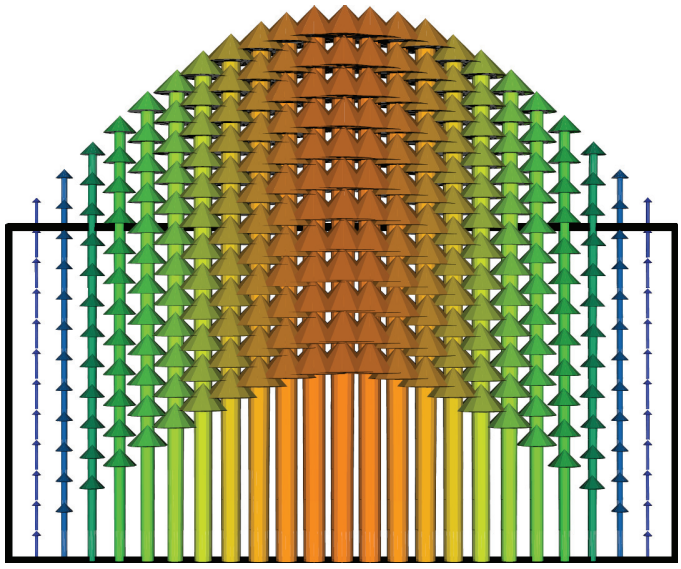

Figure 22. The TE_{10} mode for a hollow rectangular waveguide with dimensions 1.0 m × 0.5 m, computed and plotted using *DOLFIN*.

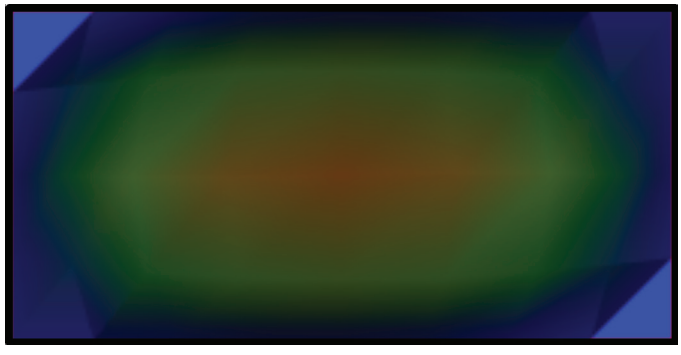

Figure 23. The axial component of the TM_{11} mode for a hollow rectangular waveguide with dimensions 1.0 m × 0.5 m, computed and plotted using *DOLFIN*.

5.3 Implementing the Source

The source implementation required us to access the internals of *DOLFIN*, since the high-level point-source class currently does not support Nédélec elements. Since an infinitesimal electrical dipole can be represented as an electrical-current point source, it entered the right-hand side (RHS) of Equation (26) as a Dirac-delta current function. Implementing the right-hand side was thus as simple as finding the element containing the source, evaluating each basis function supported in that element at the source location, and taking the dot product with the source current direction and magnitude. The function

```
calc_pointsource_contrib(V, source_coords,
source_value)
```

calculated the source contribution. This was equivalent to calculating

$$\int_V \vec{T} \bullet \delta\left(\vec{r} - \vec{r}_{source}\right) \vec{J}_{imp} dV \,, \tag{27}$$

where $\delta\left(\vec{r} - \vec{r}_{source}\right)$ is the spatial Dirac-delta function, and $\vec{J}_{imp}$ is a constant vector.

The index of the element (*FEniCS* cell) that physically contained the source was located using the mesh method `intersected_cell`. The mesh object was also attached to the function-space object, obviating the need to pass it in to our `calc_pointsource_contrib()` function (`cell_index = V.mesh().intersected_cell()`). Note that our numerically specified source coordinate had to be converted to a `dolfin.Point` object before it could be passed to `intersected_cell()`.

Using the element index thus obtained, a *DOLFIN* mesh Cell (i.e., element) object was instantiated as `c = dolfin.Cell(V.mesh(), cell_index)`. The Cell object represented the geometrical parameters of the mesh cell (e.g., its node coordinates). This object was later passed to the `dofmap` object, and was also used to evaluate the basis functions on the correct mesh cell.

Since the lower level *FEniCS* routines require pre-allocated memory space of the right type and size, we first needed to obtain the number of basis functions. This, along with methods to evaluate the basis functions, was provided by the `dolfin_element` object attached to the function space (`finite_element = V.dolfin_element()`). Since *FEniCS* supports general tensor element types, it specifies both a rank, and value dimension for each rank. Since we (should) have been dealing with standard vector elements, we first ascertained that the rank (`finit_element.value_rank()`) was 1. The value dimension of the first rank (`finite_element.value_dimension(0)`) was the dimension of the vector basis functions: it is 3 for three-dimensional Nédélec elements. The `dofnos` and `el_basis_vals` arrays were allocated using this size information, and were respectively used to store the global degree-of-freedom numbers, and the corresponding basis-function values at the source point.

The degree-of-freedom mapping object available as a member of the function-space object (`dm = V.dofmap()`) maps between the local elemental basis functions and the global degree-of-freedom numbers. The global numbers of the elemental basis functions were output to `dofnos` by passing the *DOLFIN* Cell object (`c`) that we created earlier to the `dofmap tabluate_dofs()` method (`dm.tabulate_dofs(dofnos,c)`). Each basis function was evaluated at the source point using `finite_element.evaluate_basis_all(el_basis_vals,source_coords, c)`. The array `el_basis_vals` then contained the vector-valued numerical basis-function evaluations. All that was left to do was to take the dot product with the source current using the *NumPy* `sum()` function (see Figure 25).

5.4 Implementing the NTFF Transformation

The far-field characteristics of an object are often required, e.g., for antenna and radar applications. Since the FEM can only calculate fields in finite regions of space, far-field behavior cannot be directly determined. One approach might be to model a large enough region of free space around the object to approximate far-field behavior. Apart from accuracy concerns, such an approach would have impractical memory and CPU requirements. Using a near-to-far-field (NTFF) transform, far-field behavior can be determined from near-field calculations [18].

By the surface equivalence theorem [19], knowledge of the tangential E and H fields on a closed surface containing all the inhomogeneities (i.e., the object) and all sources is sufficient to calculate the radiated field for the whole exterior region. The tangential E and H fields can respectively be seen as fictitious equivalent magnetic- and electric-current sources on the closed surface. The magnetic-current source on the surface is given by Equation (28), while the electric-current source is given by Equation (29):

$$\vec{M}_s = -\hat{n} \times \vec{E}_s \,, \tag{28}$$

$$\vec{J}_s = \hat{n} \times \vec{H}_s \,. \tag{29}$$

The E or H field at an arbitrary point can be related to the equivalent sources by the auxiliary surface magnetic and electric potential integrals, given by Equations (30) and (31), respectively:

$$\vec{A} = \frac{\mu}{4\pi} \iiint_V \vec{J}_s \frac{e^{-jkR}}{R} dv \approx \frac{\mu\, e^{-jkR}}{4\pi R} \vec{N} \,, \tag{30}$$

$$\vec{F} = \frac{\varepsilon}{4\pi} \iiint_V \vec{M}_s \frac{e^{-jkR}}{R} dv \approx \frac{\varepsilon\, e^{-jkR}}{4\pi R} \vec{L} \,. \tag{31}$$

In Equations (30) and (31), R is the distance between any point in the source and an observation point. Furthermore, ε and μ are the permittivity and permeability, such that $k^2 = \omega^2 \varepsilon \mu$. In turn, the corresponding electric and magnetic fields are determined: $\vec{E}_A$, $\vec{H}_A$ due to $\vec{A}$; and $\vec{E}_F$, $\vec{H}_F$ due to $\vec{F}$. The total fields are then obtained by the superposition of the individual fields due to $\vec{A}$ and $\vec{F}$ ($\vec{J}_S$ and $\vec{M}_S$).

The far-field behavior can be obtained by making the standard far-field approximations, resulting in the far-field potentials $\vec{N}$ and $\vec{L}$, given in Equations (32) and (33), respectively [19]:

$$\vec{N} = \int_S \vec{J}_s e^{jk_0 \vec{r}' \bullet \hat{r}} dS \,, \tag{32}$$

$$\vec{L} = \int_S \vec{M}_s e^{jk_0 \vec{r}' \bullet \hat{r}} dS \,. \tag{33}$$

The far-field observation direction is given by the observation point unit vector, $\hat{r} = \sin\theta\cos\varphi\,\hat{x} + \sin\theta\sin\varphi\,\hat{y} + \cos\theta\,\hat{z}$, while $\vec{r}'$ is the integration coordinate. The $1/R$ term, as in Equations (30) and (31), has been factored out of the integrand. After substituting the relationship for $\hat{r}$, an expression that varies purely in terms of the observation angles, θ, φ, and the integration variable, $\vec{r}'$, is obtained. To extract the θ and φ components of $\vec{N}$ and $\vec{L}$, we first need to define the unit vectors, as follows:

$$\hat{\theta} = \cos\theta\cos\varphi\,\hat{x} + \cos\theta\sin\varphi\,\hat{y} - \sin\theta\,\hat{z} \,, \tag{34}$$

$$\hat{\varphi} = -\sin\varphi\,\hat{x} + \cos\varphi\,\hat{y} \,. \tag{35}$$

The θ and φ components of $\vec{N}$ and $\vec{L}$ are given by Equations (36) to (39):

$$N_\theta = \vec{N} \bullet \hat{\theta} \,, \tag{36}$$

$$N_\varphi = \vec{N} \bullet \hat{\varphi} \,, \tag{37}$$

$$L_\theta = \vec{L} \bullet \hat{\theta} \,, \tag{38}$$

$$L_\varphi = \vec{L} \bullet \hat{\varphi} \,. \tag{39}$$

The E and H far-fields can be obtained from the potentials as [19, 20]

$$E_\theta^\infty = -\frac{jke^{-jkr}}{4\pi r}\left(L_\varphi + Z_0 N_\theta\right), \tag{40a}$$

$$E_\varphi^\infty = \frac{jke^{-jkr}}{4\pi r}\left(L_\theta - Z_0 N_\varphi\right), \tag{40b}$$

and

$$H_\theta^\infty = \frac{jke^{-jkr}}{4\pi r}\left(N_\theta - Z_0^{-1} L_\theta\right), \tag{41a}$$

$$H_\varphi^\infty = -\frac{jke^{-jkr}}{4\pi r}\left(N_\theta + Z_0^{-1} L_\varphi\right). \tag{41b}$$

Note from Equations (32) and (33) that the integrand changes for every observation angle. The full surface integration has to be reevaluated at every observation point. While far-field information is typically only needed at a limited number of points, the computational effort involved in a near-to-far-field transform can be considerable. Hence, efficient numerical implementation is desired. The *FEniCS* forms language is complete enough to express the near-to-far-field transform in terms of compilable forms expressions. The form expressions are compiled and optimized only once, but can but reused to evaluate the far field at several observation points.

```
# Define function space
V = dolfin.FunctionSpace(mesh, "Nedelec 1st kind H(curl)", order)
# Freespace wave number
k_0 = 2*np.pi*freq/c0
# Definite test- and trial functions
u = dolfin.TrialFunction(V)
v = dolfin.TestFunction(V)
# Define the bilinear forms
# Mass form
m = eps_r*inner(v, u)*dx
# Stiffness form
s = (1/mu_r)*dot(curl(v), curl(u))*dx
# Get the surface normal
n = V.cell().n
# ABC boundary condition form
s_0 = inner(cross(n, v), cross(n, u))*ds
```

Figure 24. The definitions of the function space, test and trial functions, as well as the bilinear forms.

Complications are introduced by the fact that forms in *FEniCS* can at present only express real values. This reflects the fields for which *FEniCS* was originally primarily developed for, and is at present an issue to consider before adopting it in an EM environment. To work around this, the expressions are split into real and imaginary expressions in terms of the real and imaginary E- and H-field components. This includes the real and imaginary components of the magnetic and electric current sources, such that:

$$\vec{M}_r = -\hat{n} \times \vec{E}_r \,, \tag{42a}$$

and

$$\vec{M}_i = -\hat{n} \times \vec{E}_i \,, \tag{42b}$$

with

$$\vec{J}_r = \hat{n} \times \vec{H}_r \,, \tag{43a}$$

and

$$\vec{J}_i = \hat{n} \times \vec{H}_i \,. \tag{43b}$$

The real and imaginary components of the magnetic field are then given by Equations (44a) and (44b):

$$\vec{H}_r = \frac{-\left(\nabla \times \vec{E}_i\right)}{k_0 Z_0} \tag{44a}$$

and

$$\vec{H}_i = \frac{\left(\nabla \times \vec{E}_r\right)}{k_0 Z_0} \,. \tag{44b}$$

```
def calc_pointsource_contrib(V, source_coords, source_value):
"""Calculate the RHS contribution of a current point source (i.e. electric dipole)
Input Values
-------------
@param V: dolfin FunctionSpace object
@param source_coords: length 3 array with x,y,z coordinates of point source
@param source_value: length 3 array with x,y,z components of source current
Return Values
-------------
(dofnos, rhs_contribs) with
dofnos -- Array of degree of freedom indices of the source contribution
rhs_contribs -- Numerical values of RHS contribution, such that
RHS[dofnos] += 1j*k_0*Z0*rhs_contribs will add the current
source to the system.
"""
source_coords = np.asarray(source_coords, dtype=np.float64)
source_value = np.asarray(source_value)
dm = V.dofmap()
dofnos = np.zeros(dm.max_cell_dimension(), dtype=np.uint32)
source_pt = dolfin.Point(*source_coords)
cell_index = V.mesh().any_intersected_entity(source_pt)
c = dolfin.Cell(V.mesh(), cell_index)
# Check that the source point is in this element
assert(c.intersects_exactly(source_pt))
dm.tabulate_dofs(dofnos, c)
finite_element = V.dolfin_element()
no_basis_fns = finite_element.space_dimension()
# Vector valued elements have rank of 1
assert(finite_element.value_rank() == 1)
# Vector valued elements have only one rank (i.e. 0) along which
# dimensions are defined. This is the dimension that the basis
# function value vector is. Since we have 3D Nedelec elements here
# this should be 3
bf_value_dimension = finite_element.value_dimension(0)
el_basis_vals = np.zeros((no_basis_fns, bf_value_dimension), dtype=np.float64)
finite_element.evaluate_basis_all(el_basis_vals, source_coords, c)
# Compute dot product of basis function values and source value
rhs_contribs = np.sum(el_basis_vals*source_value, axis=1)
return dofnos, rhs_contribs
```

Figure 25. The global numbers of the elemental basis functions were output to `dofnos`. Each basis function was evaluated at the source point, and the array `el_basis_vals` (then containing the vector-valued numerical basis-function evaluations) was taken as the dot product with the source current.

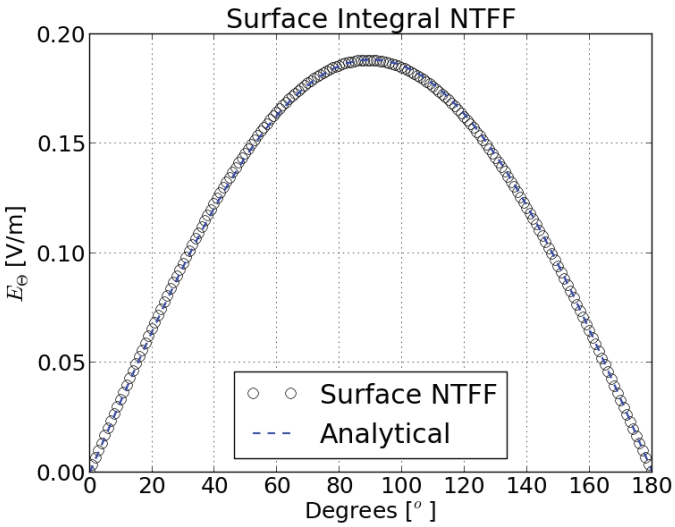

Figure 26. The results of the surface integral NTFF transformation compared to the analytical expression implemented in *SUCEM:FEM*.

The exponential representation was converted to rectangular coordinates using $e^{j\omega} = \cos\omega + j\sin\omega$. The real and imaginary components of the far-field potentials are given by

$$\vec{N}_r = \int J_r \cos\left(k_0 \vec{r}' \bullet \hat{r}\right) - J_i \sin\left(k_0 \vec{r}' \bullet \hat{r}\right) dS\left(\vec{r}'\right), \quad (45a)$$

$$\vec{N}_i = \int J_r \sin\left(k_0 \vec{r}' \bullet \hat{r}\right) + J_i \cos\left(k_0 \vec{r}' \bullet \hat{r}\right) dS\left(\vec{r}'\right), \quad (45b)$$

and

$$\vec{L}_r = \int M_r \cos\left(k_0 \vec{r}' \bullet \hat{r}\right) - M_i \sin\left(k_0 \vec{r}' \bullet \hat{r}\right) dS\left(\vec{r}'\right), \quad (46a)$$

$$\vec{L}_i = \int M_r \sin\left(k_0 \vec{r}' \bullet \hat{r}\right) + M_i \cos\left(k_0 \vec{r}' \bullet \hat{r}\right) dS\left(\vec{r}'\right). \quad (46b)$$

A further complication is that the potentials directly integrate vector quantities. Vector forms expressions are allowed, but the end result must be a scalar. This can be solved by dotting the vector quantity by the theta ($\hat{\theta}$) and phi ($\hat{\varphi}$) unit vectors, as given in Equations (36) to (39).

The resulting expressions are somewhat lengthy, but reasonably straightforward. The reader is invited to peruse the definition of the `ntff()` function in the source code. The code is well commented, and its meaning should be clear, given the above description. The surface integral near-to-far-field (NTFF) transformation was implemented in *FEniCS* using *SUCEM:FEM*, and the results (Figure 26) were compared to the analytical far-field expression for an infinitesimal dipole given in Equation (47):

$$\vec{E}_\theta = jZ_0 k_0 \lambda \frac{e^{-jk_0 r}}{4\pi r} \sin\theta. \quad (47)$$

6. Conclusion

The use of the *Python* front end of *DOLFIN*, *PyDOLFIN*, to model various EM problems has been investigated. Some elementary problems were implemented in *FEniCS*, including the scalar potential solution to closed- and open-boundary electrostatic problems, as well as the cutoff and dispersion analysis of a hollow rectangular waveguide. More advanced topics considered were the implementation of an infinitesimal dipole and the near-field-to-far-field transformations for the FEM finite regions in space.

As with many packages developed in other fields of computational science and engineering, complex arithmetic is unfortunately not currently supported in *FEniCS*. (Of course, for time-domain FEM this is not an issue, as such codes use only real-valued operations). Vector elements are supported, which is often *not* the case with FEM packages originating outside electromagnetics. Notwithstanding the lack of support for complex arithmetic, the relative ease with which basic FEM operations can be performed, and the support for vector elements, make *FEniCS* an attractive environment for prototyping finite-element codes in electromagnetics.

7. References

1. "The FEniCS Project," available at http://www.fenicsproject.org/.

2. D. B. Davidson, "Implementation Issues for Three-Dimensional Vector FEM Programs," *IEEE Antennas and Propagation Magazine*, **42**, 6, December, 2000, pp. 100-107.

3. A. Awadhiya, P. Barba, and L. Kempel, "Finite Element Method Programming Made Easy," *IEEE Antennas and Propagation Magazine*, **45**, 4, August 2000, pp. 73-79.

4. J. C. Nédélec, "Mixed Finite Elements in $\Re^3$," *Numerische Mathematik*, **35**, 1980, pp. 315-341.

5. J. C. Nédélec, "A New Family of Mixed Finite Elements in $\Re^3$," *Numerische Mathematik*, **35**, 1980, pp. 315-341.

6. P. P. Silvester and R. L. Ferrari, *Finite Elements for Electrical Engineers, Third Edition*, Cambridge, UK, Cambridge University Press, 1996.

7. J. L. Volakis, A. Chatterjee and L. C. Kempel, *Finite Element Method for Electromagnetics: Antennas, Microwave Circuits and Scattering Applications*, Oxford, UK, Oxford University Press and IEEE Press, 1998.

8. "Gmsh: A Three-Dimensional Finite Element Mesh Generator with Built-In Pre- and Post-Processing Facilities," available at http://www.geuz.org/gmsh/.

9. J. Jin, *The Finite Element Method in Electromagnetics, Second Edition*, New York, Wiley, 2002.

10. A. Logg, K.-A. Mardal, and G. Wells (eds.), *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, Berlin, Springer, 2012.

11. D. B. Davidson, *Computational Electromagnetics for RF and Microwave Engineers, Second Edition*, Cambridge, UK, Cambridge University Press, 2011.

12. ParaView, available at http://www.paraview.org/.

13. G. S. Smith, *An Introduction to Classical Electromagnetic Radiation*, Cambridge, Cambridge University Press, 1997.

14. G. Pelosi, R. Coccioli, and S. Selleri, *Quick Finite Elements for Electromagnetic Waves*, Norwood, MA, Artech House, 1998.

15. D. M. Pozar, *Microwave Engineering, Third Edition*, New York, Wiley, 2005.

16. SUCEM:FEM, Stellenbosch University Computational Electromagnetics: Finite Element Method, available at https://github.com/cemagg/sucem-fem.

17. A. J. Otto, E. Lezar, N. Marais, R. G. Marchand and D. B. Davidson, "Rapid, High-Order Finite Element Modelling with FEniCS and SUCEM:FEM," 11th International Workshop on Finite Elements for Microwave Engineering (FEM2012), Estes Park, Colorado, USA, June 4-6, 2012.

18. P. Monk, "The Near Field to Far Field Transformation," *The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, **14**, 1, 1995, pp. 41-56.

19. C. A. Balanis, *Antenna Theory: Analysis and Design, Third Edition,* New York, Wiley, 2005.

20. C. A. Balanis, *Advanced Engineering Electromagnetics, Second Edition*, New York, Wiley, 2012.