An Open-Source Cross-Platform Finite-Element Modeling Program

EECE249-Numerical Techniques in Electromagnetics

Jason Sidabras

April 28, 2006

1 Introduction

Finite-Element Modeling (FEM) is a numerical method that discretizes a large problem into a group of coupled, more manageable problems in order to solve them. These groups are formed through cells, called elements, and are coupled in a mesh. An example of a mesh cell and its node numbering scheme can be seen in Fig. 1. This paper will assume triangular mesh elements will be used. Each node could either have a fixed potential or that potential is what is needed to be solved. A node with a known potential is called a prescribed node, while a node without a known potential is called a free node. It is the free nodes, of the relationship to the free nodes to the prescribed nodes that Finite-Element Modeling tries to solve.

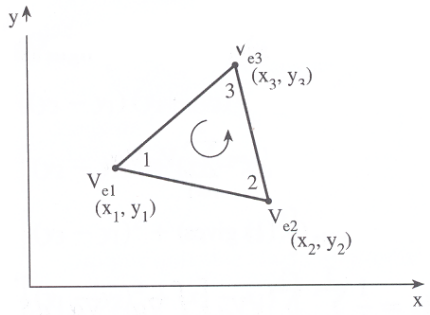

Figure 1: Finite element and its local numbering scheme.[3]

With commercial products such as Ansoft HFSS and Computer Simulated Technology's Microwave Studio, the FEM market seems to have good representation. The problem is these programs are closed source and do not lend themselves easily to students looking for inner workings of how the finite-element analysis technique calculates the numerical solution. This project aims to give a fundamental background in FEM and allows for other students to easily learn the basics of finite-element numerical techniques. The focus of this paper will be calculating the cross-sectional wavenumber for various geometries.

2 Theory

Finite-element modeling numerical technique can be be summarized in four basic steps[2] [1]:

1. Discretization or subdivision of the domain
2. Selection of the interpolation functions
3. Formulation of the system of equations
4. Solution of the system of equations

There are two methods of solving a finite-element problem. The first is an iterative method and the second is a discrete method. The iterative method uses a system of equations that have to be solved in series, where the answer to the last set becomes the starting point of the next. The discrete method uses a system of equations that are solved at one time. One example, and the focus of this paper, is the eigenvalue problem.

2.1 Eigenvalue problems

The eigenvalue problem assumes that both the differential equations and the boundary conditions are homogeneous. It can be thought of that there is no source exciting the mode of interest. This becomes very

important in geometries where the modes of the system are not known. The system of equations used for the eigenvalue problem can be generalized in:

$$\left[\ A\ \right]\left[\ \phi\ \right] = \lambda \left[\ B\ \right]\left[\ \phi\ \right] \tag{1}$$

where **A** and **B** are known matrices and λ and ϕ are the unknowns. Solving for λ, the eigenvalue, gives a distinct ϕ, the eigenvector. Solving for the eigenvalue is done by solving:

$$\left|\ B^{-1}A\ \ -\lambda I\right| = 0 \tag{2}$$

where **I** is the identity matrix. Solving the determinate, known as the characteristic equation, and finding its roots will yield the system's eigenvalues. From that the eigenvectors could be solved for through multiple linear algebraic techniques.

2.2 Formulation of the system equation

In this paper the system equation relates to the construction of the functional from a partial differential equation (PDE). In the case of the cross-section of a waveguide, the PDE is the wave equation:

$$\nabla^2\phi + k^2\phi = 0 \tag{3}$$

Using the following steps the functional can be derived[3].

- Multiply the operator equation $L\phi = 0$ with the variational $\delta\phi$
- Use integration by parts to transfer the derivatives to variation $\delta\phi$
- Apply boundary conditions
- Bring variational operator δ our of the integral and cancel

These steps are show in full detail in Appendix E. Through these steps, the functional for the wave equation becomes:

$$I(\phi) = \frac{1}{2}\int \left[|\nabla\phi|^2 - k^2\phi^2\right] \tag{4}$$

Putting the eq. 4 in matrix form yields:

$$I(\phi) = \frac{1}{2}\left[\phi\right]^T\left[C\right]\left[\phi\right] - \frac{k^2}{2}\left[\phi\right]^t\left[T\right]\left[\phi\right] \tag{5}$$

Now this system equation can be broken up into a matrix describing each element and then a linking global matrix which describes the relationships between each element, **C** and its projection **T**. This is known as discretization of the domain.

2.3 Discretization of the domain

The first step in solving a finite-element problem is breaking up the domain into non-overlapping elements. These elements can be of any shape and size. This paper will focus on the three-node triangle pictured in Fig. 1. It is important to note the numbering of the elemental nodes in counter-clockwise order. This is done to keep the calculation of the area positive.

Area can be calculated using:

$$A = \frac{1}{2}\left[(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)\right] \tag{6}$$

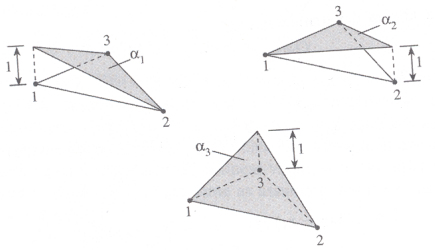

Figure 2: Finite element and its projection on itself and other nodes.[3]

Using the area one has to find the projection of the elements related to each other, or the shape function. The shape function follows the equation:

$$C_{ij}^{(e)} = \int \nabla \alpha_i \cdot \nabla \alpha_j \tag{7}$$

A visual of this function can be found in Fig. 2. Where the projection on each node is zero, unless $i = j$. Then linear segments connect the nodes. This can be done in matrix form by using:

$$P1 = (y_2 - y_3) \tag{8}$$
$$P2 = (y_3 - y_1) \tag{9}$$
$$P3 = (y_1 - y_2) \tag{10}$$
$$Q1 = (x_3 - x_2) \tag{11}$$
$$Q2 = (x_1 - x_3) \tag{12}$$
$$Q3 = (x_2 - X_1) \tag{13}$$
$$Coef = 1/(4 * Area) \tag{14}$$
$$C11 = Coef * (P1 * P1 + Q1 * Q1) \tag{15}$$
$$C22 = Coef * (P2 * P2 + Q2 * Q2) \tag{16}$$
$$C33 = Coef * (P3 * P3 + Q3 * Q3) \tag{17}$$
$$C12 = Coef * (P1 * P2 + Q1 * Q2) \tag{18}$$
$$C13 = Coef * (P1 * P3 + Q1 * Q3) \tag{19}$$
$$C23 = Coef * (P2 * P3 + Q2 * Q3) \tag{20}$$
$$C32 = C23, C31 = C13, C21 = C12 \tag{21}$$

and from this an element coefficient matrix, or "stiffness matrix" can be formed for each element (e) in the problem.

$$\left[C^{(e)}\right] = \begin{bmatrix} C_{11}^{(e)} & C_{12}^{(e)} & C_{13}^{(e)} \\ C_{21}^{(e)} & C_{22}^{(e)} & C_{23}^{(e)} \\ C_{31}^{(e)} & C_{32}^{(e)} & C_{33}^{(e)} \end{bmatrix} \tag{22}$$

Equivalently a fundamental matrix $\mathbf{T}$ can be defined as:

$$T_{ij}^{(e)} = \int \alpha_i \alpha_j d\mathbf{S} \tag{23}$$

in matrix form it can be shown that:

$$T_{ij}^{(e)} = \begin{matrix} A/12 & i \neq j \\ A/6 & i = j \end{matrix} \tag{24}$$

2.3.1 Global Matrix

The global matrix refers to either the **C** or **T** matrix. The global matrix can be defined as the matrix on how all of the element coefficient matrices are related. For example, Fig. 3 has two connected matrices. Each matrix has an 'element' node number and a 'global' node number. In this case the relationship of $C_{44} = C_{33}^1 + C_{33}^2$, because both element 1 and element 2 are connected to the global node 4 by their respective element node 3. While $C_{13} = 0$, because there is no connecting node(s) between element 1 and 2 connecting global nodes 1 and 3. A full example for a 9 node square waveguide is shown in Appendix D.

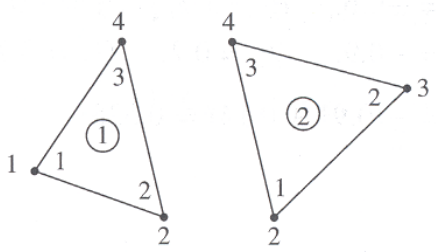

Figure 3: Two elements and their local and global node numbering.[3]

Once the global matrices are found inserting them back into the functional (eqn. 5) yeilds:

$$I(\phi) = \frac{1}{2}[\phi]^T [C][\phi] - \frac{k^2}{2}[\phi]^t [T][\phi] \tag{25}$$

A quick check to see if your global matrix was constructed correctly is to sum up each row. The sum of each row should equal zero. After the global matrices are assembled, apply the boundary conditions to get two distinct eigenmode equations:
TM Modes:

$$\left[\ C_{ff}\ \right]\left[\ \phi_{ff}\ \right] = \lambda \left[\ T_{ff}\ \right]\left[\ \phi_{ff}\ \right] \tag{26}$$

and TE Modes:

$$\left[\begin{matrix} C_{ff} & C_{fp} \end{matrix}\right]\left[\begin{matrix} \phi_{ff} \\ \phi_{fp} \end{matrix}\right] = \lambda \left[\begin{matrix} T_{ff} & T_{fp} \end{matrix}\right]\left[\begin{matrix} \phi_{ff} \\ \phi_{fp} \end{matrix}\right] \tag{27}$$

where A_{ff} denotes, freenode-freenode interconnects and A_{fp} denotes freenode-prescribed node interconnects.

3 Methods

All calculations were done on a Apple PowerBook G4 running OS X 10.4.6 with a 1.67 GHz Freescale PowerPC CPU using 512MB L2 cache. The system memory has a total of 1 GB of RAM. During calculations an average of 85% of the available CPU time was used by the program along with 7.39 MB of RAM. The current implementation of the program uses only one thread but some calculations, such as the **C** matrix and **T** matrix projections could be threaded. Also, simplifications of the global matrices could be done to speed up the solution time. This was considered beyond the focus of the project.

Several geometries were created to test the validity of both the solutions and the intermediate steps that the program was running. This aided in the debugging process. The simplest geometry to create and solve is the square waveguide. A square waveguide cross-section with 9, 12, 25, 36 and 49 nodes was created by hand to thoroughly test the **pyfem** program. The lowest order TM mode of this geometry was calculated using eq. 29 and compared to the numerical solution using eqn. 30.

$$k^2 = \left(\frac{m\pi}{a}\right)^2 + \left(\frac{n\pi}{b}\right)^2 \tag{28}$$

$$\%error = \frac{|k_{solved} - k_{exact}|}{k_{exact}} x100\% \tag{29}$$

Once acceptable answers were realized, four other geometries were solved. Fig. 4 shows the cross-sections that were solved by this program.

The rectangular shape, Fig. 4b, allowed for another exact solution of the cut-off wavenumber to be calculated using eq. 29 and compared to the **pyfem** program. The rectangular shape also was a more practical waveguide cross-section and allowed for other modes to exist in its geometry. The L-Shaped geometry, Fig. 4c, was chosen because a paper by Bernard Schiff and Zohar Yosibash solved this particular problem using their own Finite-Element Modeling program[4]. In this paper, there was a great deal of time taken to properly mesh the corners of an L-Shaped waveguide. Using their calculated data, the L-Shaped geometry allowed for the **pyfem** program to be tested with a more complicated shape. The other two shapes, Figs. 4d and e, were chosen for their complexity in analytically solving the cut-off wavenumber.

Solving the geometry was done by running the command:

```
$ time ./femtest.py meshfile.msh
```

The Unix command *time* was used to keep track of how long each problem took to solve.

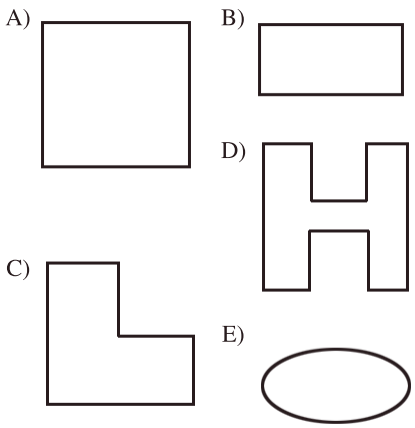

Figure 4: Five geometries were solved using the FEM program. A) Square 2 x 2 , B) Rectangular 2 x 1, C) L-Shaped 2 x 2 square with a 1 x 1 square cut out, D) Ridged Waveguide 2 x 2 with 0.5 x 0.5 gap and an E) Ellipse with a major axis of 2 and a minor axis of 1. All use arbitrary units.

3.1 Programming Language

The programming language that was chosen for this project was Python (http://www.python.org). Python is an open-source cross-platform scripting language with a wide variety of uses. Some uses include web development, software development and NASA interfaces for Space Shuttle missions. Python was chosen for this development because of its "pseudo-code" like appearance, simple syntax structure and the authors familiarity with its development. Python can easily be converted into compiled languages like FORTRAN or C++ to increase calculation speed.

Below is a sample program:

```
######
# Description: This program outputs 'Hello World!' until the loop
#     is up.
#####
i = 0
while i < 100:
   print "Hello World!"
   i = i + 1
```

Another advantage of the Python scripting language is its ability to add modules to the core code. This allows for an almost endless library of functions to be included in your program by using the *import* command. Even this program uses the flexibility of the *import* command by importing modules such as **math** and **Scipy**. The main program was written as a module class called **pyfem** to be imported in the interface program **femtest** (See Appendix A and Appendix B for the complete programs). Breaking the program into a class and an interface program has its advantages in debugging and in porting your functions to other applications.

3.1.1 Scipy Module

One Specific module used in this project was the **Scipy** module (http://www.**scipy**.org/). **Scipy** is described as "open-source software for mathematics, science, and engineering". The current version 0.4.8 and is the last stable release before a major overhaul and bug test release, due soon. **Scipy** is based on a previous module called **Numpy** and has the ability to manipulate NxN array matrices. Using **Scipy**, matrix manipulation and solving of the eigenvalues and eigenvector is made into one convenient set of commands.

3.2 Meshing Scheme

This projects main focus was the solution of the cut-off wavenumber for any given waveguide geometry. To save on time, it was decided to use an open-source and cross-platform mesh generating program called Gmsh (http://www.geuz.org/gmsh/). Gmsh is a 3-D meshing program that has a semi-intuitive OpenGL interface to draw the geometry. Since the focus of this project is 2-D waveguide cross-sections, Gmsh could be considered overkill. But because of its good documentation, intuitive ASCII '.msh' files and the ability, at a later date, to have a post-processing file displayed along with the geometry, Gmsh was chosen as the mesh generating program. Fig. 5 shows a screenshot of Gmsh in action.

The '.msh' file format has two versions. Version 1.0 can be interpreted by this program (An example mesh file can be found in Appendix C). In the '.msh' file format, version 1.0, the file is divided in two sections, defining the nodes (`$NOD-$ENDNOD`) and the elements (`$ELM-$ENDELM`) in the mesh:

```
$NOD
number-of-nodes
node-number x-coord y-coord z-coord
...
$ENDNOD
$ELM
```

```
number-of-elements
elm-number elm-type reg-phys reg-elem number-of-nodes node-number-list
...
$ENDELM
```

The **pyfem** class only parses a few of these parameters:

- *number-of-nodes* or *number-of-elements*: total number of nodes or elements respectively. Used to check to make sure the number of inputted lines match the number of nodes
- *x-coord y-coord z-coord*: placement of said node. Only the x- and y-coord are used. Since it is assumed this is a 2-D problem, the z-coord is removed during conditioning in the **pyfem** class.
- *elm-number*: number of the element.
- *number-of-nodes*: number of the nodes to create each element. Lines, used for prescribed nodes and boundary conditions, have two nodes. Triangle elements use three nodes.
- *node-number-list*: list of the nodes used for each element. Used to 'recursive link' the node number with the appropriate x- and y-coord.

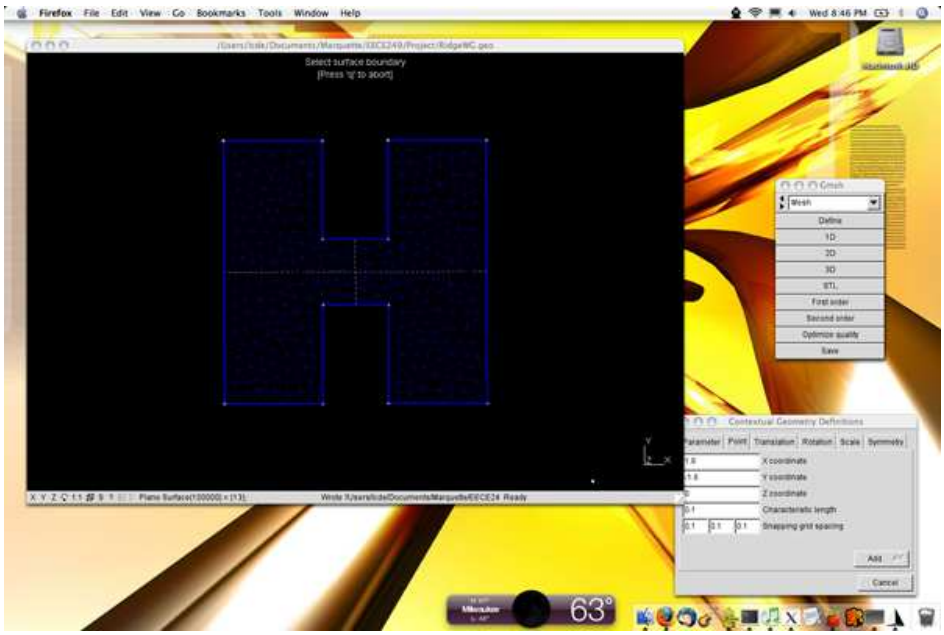

Figure 5: A typical screenshot of Gmsh with both the geometry file and the created mesh displayed.

One complication was found with using Gmsh's drawing program and meshing. It was found that if, for example, three elements were placed {1, 2, 3} and the second element was removed from the drawing using a simple erase function, Gmsh would not adjust the number of the elements. So a line that is connected by the two point elements would be {1, 3}, where point element {2} would be missing. This complication was not accounted for in the **pyfem** class and errors occurred. Many work arounds were tried with little success. It was found the best solution is to plan out the cross-section and the points you will use before drawing. If all the points are used and no points are erased, there is no complication.

4 Results

The results of this project solve for the TM wavenumber of five different cross-sections: Square Waveguide, Rectangular Waveguide, L-Shaped Waveguide, Ridged Waveguide and Elliptical Waveguide. The robustness of this program allows for any line element to be considered a metallic wall of infinite conductivity, allwing for any cross-sectional geometry to be solved.

4.1 TM Modes

Using the above theory, a calculation of the TM Modes could be simulated. In Appendix A, the **pyfem** class includes a function that uses the ϕ_{ff} nodes to create a global free node **C** and **T** matrix to solve eq. 30 for the wavenumber k, where $k = \sqrt{\lambda}$.

$$\left[\ C_{ff}\ \right]\left[\ \phi_{ff}\ \right] = \lambda \left[\ T_{ff}\ \right]\left[\ \phi_{ff}\ \right] \tag{30}$$

4.1.1 Square Waveguide

Using the eqn. 28, the cut-off wavenumber can be found for the fundamental transverse magnetic mode, TM_{11}. The results can be found in Table 1. This test geometry shows good convergence around 49 nodes.

Table 1: Table of calculated TM_{11} values and the %error of each value using a certain set of nodes for the Square Waveguide.

nodes	k-value	%error	time
9	2.44948974278	10.266	0m0.796s
12	2.76339711883	24.397	0m0.830s
25	2.66369041561	19.908	0m0.872s
36	2.51901893031	13.396	0m0.883s
49	2.29764827250	3.431	0m4.168s
495	2.22533974726	0.175	75m27.641s

An additional solution meshed by Gmsh using 495 nodes was also calculated. With the increased number of nodes the solution time was over an hour, but the error was below 1%.

4.1.2 Rectangular Waveguide

As a second sanity check, and for a more practical application a rectangular waveguide cross-section was calculated. This cross-section was checked for the first two fundamental modes. Good results were found, and it was assumed that the rest of the wavenumbers matched the analytical solution.

Table 2: Table of first six calculated TM cut-off wavenumber values for the Rectangle Waveguide.

nodes	k-value	time	Mode	%error
250	3.53729457818	8m48.330s	TM_{11}	0.70
-	4.50277280315	-	TM_{21}	1.35
-	5.76978686264	-		
-	6.58889892046	-		
-	7.15305730281	-		
-	7.24565757325	-		

4.1.3 L-Shaped Waveguide

The **pyfem** program was compared to Schiff *et el.*'s FEM program. Their program, abbreviate here as IEEE, took special care to use meshes specifically designed for waveguide cross-sections with corners. Some examples that they did were an L-Shaped Waveguide and a very reentrant ridged waveguide structure. In this paper we have compared the solutions found using the IEEE program to the solutions using **pyfem**. Assuming IEEE solutions are absolute truth, the agreement was exceptional. **Pyfem** used 380 nodes and solved in about 31 minutes. The data is found in Table 2.

Table 3: Table of first six calculated TM cut-off wavenumber values for the L-Shaped Waveguide compared to the IEEE FEM program.

pyfem	IEEE	%error	time
3.12956940458	3.1069	0.73	31m34.088s
3.91888315457	3.9023	0.42	-
4.47302862948	4.4492	0.54	-
5.48922248335	5.4540	0.65	-
5.72920865352	5.6742	0.97	-

Some advantages of **pyfem** over IEEE is the significant digits. Using an iterative method, IEEE was only accurate to four places after the decimal point. With **pyfem**, a direct approach was used allowing for the significant digits to be as large as the machine code allows. Some disadvantages may include the amount of time it takes to solve the problem. At no point does **pyfem** use any optimization or simplification of matrix data.

4.1.4 Other Geometries

The following solved wavenumber values were not checked by any analytical solution. It was assumed because the above results that these wavenumber values were within one percent and had adequate meshing.

The complexity of the ridged waveguide made for an increase in the number of nodes. This increase had a direct correlation to the amount of time to solve the problem. It should be noted that ridged waveguide is not normally used in the TM propagating case. Normally a fundamental TE_{01} mode is used to propagate down the ridged waveguide creating a distinct lumped-circuit inductance and capacitance in a distributed model. The wavenumber is then easily estimated by:

$$k = \frac{\sqrt{\mu\epsilon}}{\sqrt{LC}} \tag{31}$$

In the case of the elliptical waveguide, complex elliptical integrals must be solved. Normally these are solved by numerical techniques, such as the one described in this paper. The first six TM cut-off wavenumbers for the ridged waveguide and elliptical waveguide can be found in Table 4.

Table 4: Table of first six calculated TM cut-off wavenumber values for the Ridged Waveguide (left) and Elliptical Waveguide(right).

nodes	k-value	time	nodes	k-value	time
426	4.26073959057	44m4.883s	205	3.7968633252	4m51.121s
-	4.30627842965	-	-	5.05516106215	-
-	5.27768727592	-	-	6.42031044828	-
-	5.32883111201	-	-	6.98041362081	-
-	6.16254909823	-	-	7.87050872983	-
-	6.30430585383	-	-	8.17335476558	-

4.2 TE Modes

Using the above theory a calculation of the TE Modes could be simulated. In Appendix A, the **pyfem** class includes a function that should use both the ϕ_{ff} and ϕ_{fp} modes to create a global matrix to solve the equation:

$$\begin{bmatrix} C_{ff} & C_{fp} \end{bmatrix} \begin{bmatrix} \phi_{ff} \\ \phi_{fp} \end{bmatrix} = \lambda \begin{bmatrix} T_{ff} & T_{fp} \end{bmatrix} \begin{bmatrix} \phi_{ff} \\ \phi_{fp} \end{bmatrix} \tag{32}$$

At the time of writing this paper, this function does not work. It is believed that the setup of the free to prescribed nodes in the matrix are not correctly accounted for.

5 Conclusion

It has been shown in this paper, that a practical Finite-Element Modeling program with a low error percentage can be calculated using open-source cross-platform solutions. The accuracy of this program is comparable to that of other home-built specialized FEM code. With more time a TE cut-off wavenumber solution can be calculated, along with a full vector analysis of the TE and TM fields. This project has successfully taught the author the fundamentals of the FEM numerical technique and has given the author further insight into the complexity of commercial software.

References

[1] R. L. Coren. Basic engineering electromagnetics; an applied approach. 1989.

[2] J. Jin. The finite element method in electromagnetics, second edition. 2002.

[3] M. N. O. Sadiku. Numerical techniques in electromagnetics, second edition. 2001.

[4] B. Schiff and Z. Yosibash. Eigenvalues for waveguides containing re-entrant corners by a finite-element method with superelements. *IEEE Transactions on Microwave Theory and Techniques*, 48:291, 2000.

A PyFEM Class

Below is the main class of the finite-element modeling program. Each function is described in detail within the commenting of the code. Appendix C demonstrates the program interface.

```
#!/usr/local/bin/python
######################
#  filename: pyfem.py
#  written by: Jason Sidabras
#  Description: This file is the main pyfem class
#  Requirements: Python 2.4
#           gmsh 1.63: msh version 1 output
#           {\bf scipy} 0.9.6
#  Usage: import pyfem
######################
import {\bf scipy}
import os.path
import re
import math

femVersion = '0.1'
##################################
# nodes(filename)
# Description:  grabs all nodes and returns complete list of nodes
# Output: returns new list
#
##################################
def nodes(filename):
    file = open(filename, 'r')
    count_lines = len(file.readlines())
    file = open(filename, 'r')
    check = 0
    loop = 0
    nodeList= []
    while count_lines != check:
        header = file.readline()
        header = header.strip()
        if header == "$NOD":
            numnodes = file.readline()
            numnodes = numnodes.strip()
            numnodes = int(numnodes)
            while loop < numnodes:
                loop = int(loop)
                test2 = file.readline()
                test2 = test2.strip()
                if test2 == '$ENDNOD':
                    print "break"
                    break
                else:
                    newList = nodematrix(test2)
                    nodeList.append(newList)
                loop = loop + 1
```

```
        check = check + 1
    return nodeList

##################################
# nodematrix(list)
# Description:  removes z component and conditions matrix to accept floats
# Output: returns new list
#
##################################
def nodematrix(list):
    newList = []
    newList = list.split()
    lenth = len(newList)
    newList[0]=int(newList[0])
    loop = 1
    while loop < lenth:
        newList[loop] = float(newList[loop])
        loop = loop + 1
    #remove for z component
    newList.pop()
    return newList


##################################
# Elms(filename, 1 or 2)
# Description:  This function takes the file and returns either:
#   1:  line elements associated with boundaries  2 nodes
#   2:  Triangle elements 3 nodes
# Output: returns new list
#
##################################
def Elms(filename, number):
    file = open(filename, 'r')
    count_lines = len(file.readlines())
    file = open(filename, 'r')
    check = 0
    loop = 0
    elmList= []
    while count_lines != check:
        header = file.readline()
        header = header.strip()
        if header == "$ELM":
            numnodes = file.readline()
            numnodes = numnodes.strip()
            numnodes = int(numnodes)
            while loop < numnodes:
                loop = int(loop)
                test2 = file.readline()
                test2 = test2.strip()
                if test2 == '$ENDELM':
                    print "Reached End for Some Reason."
```

```
                break
            else:
                newList = elmmatrix(test2)
                if newList[0] == int(number):
                    newList.pop(0)
                    newList.pop(0)
                    newList.pop(0)
                    newList.pop(0)
                    elmList.append(newList)
            loop = loop + 1
        check = check + 1
    return elmList

def elmmatrix(list):
    newList = []
    newList = list.split()
    lenth = len(newList)
    newList[0]=int(newList[0])
    loop = 1
    while loop < lenth:
        newList[loop] = int(newList[loop])
        loop = loop + 1
    newList.pop(0)
    return newList

##################################
# elmRecurse(matrix)
# Description:  This function takes the element list and the node list and returns
#    a new list where all of the nodes have been replaced with x-y coords
# Output: return new matrix
#
##################################
def elmRecurse(elementlist, nodelist):
    if elementlist == None:
        print "ERROR"
        return None
    check = 0
    elmLength = len(elementlist)
    numberofEls = len(elementlist[0])
    while check < elmLength:
        check3 = 0
        while check3 < numberofEls:
            #Search for Node number
            #get working number
            nodeNumber = elementlist[check][check3]
            #how many times to cycle nodelist
            nodeLength = len(nodelist)
            check2 = 0
            while check2 < nodeLength:
                if nodeNumber == nodelist[check2][0]:
                    step = nodelist[check2]
```

```
                elementlist[check][check3] = step
                break
            check2 = check2 + 1
        check3 = check3 + 1
    check = check + 1
elementlist = trunk(elementlist)
return elementlist

##################################
# trunk(matrix)
# Description:  trunkate the element number out of the matrix, leaving
#    only the x and y coords.
# Output: Return new matrix
#
##################################
def trunk(matrix):
    newList = []
    newList2 = []
    length = len(matrix) - 1
    length2 = len(matrix[0][0]) - 1
    # dive deep into the matrix until you get the single [num, x, y] vector
    # grab the length of that vector and make sure its 3, if its 2 ([x , y]) skip
    # that means you've already did it.
    # kind of a hack but it seems to work
    for xy in matrix:
        for y in xy:
            loop = 0
            leng = len(y)
            if leng == 3:
                y.pop(0)
    return matrix

##################################
# elementCoeff(Elementlist)
# Description: Create element projection matrix C
# Output: Return projection C Matrix
#
##################################
def elementCoeff(elementlist):
    ElementCoeffMatrix=[]
    for element in elementlist:
        # in ELEMENT ->
        # ELEMENT[0] is first element
        # ELEMENT[0][0] is first element X
        # ELEMENT[0][1] is first element Y
        one = 0
        two = 1
        three = 2
        x = 0
        y = 1
        P1 = (element[two][y] - element[three][y])
```

```
        P2 = (element[three][y] - element[one][y])
        P3 = (element[one][y] - element[two][y])
        Q1 = (element[three][x] - element[two][x])
        Q2 = (element[one][x] - element[three][x])
        Q3 = (element[two][x] - element[one][x])
        Area = 0.5*(P2*Q3 - P3*Q2)
        Coef = 1/(4*Area)
        C11 = Coef*(P1*P1 + Q1*Q1)
        C22 = Coef*(P2*P2 + Q2*Q2)
        C33 = Coef*(P3*P3 + Q3*Q3)
        C12 = Coef*(P1*P2 + Q1*Q2)
        C21 = C12
        C13 = Coef*(P1*P3 + Q1*Q3)
        C31 = C13
        C23 = Coef*(P2*P3 + Q2*Q3)
        C32 = C23
        CoeffMatrix = [[C11, C12, C13],[C21, C22, C23],[C31,C32,C33]]

        ElementCoeffMatrix.append(CoeffMatrix)
    return ElementCoeffMatrix

################################
# GlobalMatrix(ElementCoeffMatrix, filename)
# Description: Create general Global matrix
# Output: Return Global matrix
#
################################
def GlobalMatrix(elementCoeffMatrix, filename):
    # Nodes x Nodes sized matrix
    # Sum i 1-N Cij = 0 = Sum j 1-N Cij
    Elements = Elms(filename,2)
    node = nodes(filename)
    nodelen = len(node)
    GlobalMatrix = []
    # Find Golbal Cij elements
    i = 1
    while i <= nodelen:
        GlobalElementList = []
        j = 1

        while j <= nodelen:
            GlobalElement = 0
            elementnumber = 1
            if i == j:
                for elem in Elements:
                    nodenumber = 0
                    for node in elem:
                        if node == j:
                            GlobalElement = GlobalElement + \
                    elementCoeffMatrix[elementnumber - 1][nodenumber][nodenumber]
                        nodenumber = nodenumber + 1
```

```
                elementnumber = elementnumber + 1
            # find the rest
            else:
                for elem in Elements:
                    nodenumber2 = 0
                    for nodej in elem:
                        nodenumber = 0
                        for nodei in elem:
                            if nodei == i:
                                if nodej == j:
                                    GlobalElement = GlobalElement + \
                        elementCoeffMatrix[elementnumber - 1][nodenumber][nodenumber2]
                                nodenumber2 = nodenumber2 + 1
                            nodenumber = nodenumber + 1
                    elementnumber = elementnumber + 1
            GlobalElementList.append(GlobalElement)
            j = j + 1
        GlobalMatrix.append(GlobalElementList)
        i = i + 1
    return GlobalMatrix

##################################
# TMatrix(elementlist, filenam)
# Description: This function creates an elemental matrix then creates a global T matrix
# Output: Global T Matrix
#
##################################
def TMSolve(C, T, Number):
    NewCMatrix = []
    NewTMatrix = []
    nodelen = len(Number)
    i = 1
    Num = 0
    while i < nodelen:
        TList = []
        CList = []
        j = 1
        while j < nodelen:
            #Ftt = 'T' + str(Num+i-1) + '_' + str(Num+j-1)
            Ftt = T[Number[i]-1][Number[j]-1]
            TList.append(Ftt)
            #Fcc = 'C' + str(Num+i-1) + '_' + str(Num+j-1)
            Fcc = C[Number[i]-1][Number[j]-1]
            CList.append(Fcc)
            j = j + 1
        i = i + 1
        NewTMatrix.append(TList)
        NewCMatrix.append(CList)
    Tff = {\bf scipy}.mat(NewTMatrix)
    Cff = {\bf scipy}.mat(NewCMatrix)
    TffI = Tff.I
```

```
    AMat = TffI*Cff
    lam = {\bf scipy}.linalg.eigvals(AMat)
    return lam

##################################
# FreeNode(ElementCoeffMatrix, ElementList, filename)
# Description: This function finds the free nodes in the node list
# Output: free node list
#
##################################
def FreeNode(elementCoeffMatrix, elementlist, filename):
    C = GlobalMatrix(elementCoeffMatrix, filename)
    T = TMatrix(elementlist, filename)
    BoarderElms = Elms(filename, 1)

    NewMatrix = []
    NumberT = []
    NumberT2 = []
    NumberC = []
    NumberC2 = []
    loop = 0
    nodelist = nodes(filename)
    length = len(nodelist)
    for q in nodelist:
        NumberT.append(q[0])
    lf = []
    PrescribedNodeFlag = 0;
    length3 = length - len(BoarderElms)
    for i in range(length):
        Cff = []
        Tff = []
        for k in range(len(BoarderElms)+1):
            if len(NumberT) == length3:
                break
            if i == (BoarderElms[k][0]):
                PrescribedNodeFlag = 1
                break
            if PrescribedNodeFlag == 0:
                p = BoarderElms[k][0]
                if NumberT.count( p ) != 0:
                    NumberT.remove(p)
                else:
                    break
            else:
                PrescribeNodeFlag = 0
    return NumberT

##################################
# TMSolve(C, T, FreeNodes)
# Description: This function takes a global T and C matrix and
#   along with a list of FreeNodes and solves the needed matrix
```

```
#   for the TM mode case.
# Output: lambda values, lambda = k^2
#
#################################

def TMSolve(C, T, Number):
    NewCMatrix = []
    NewTMatrix = []
    Cff = {\bf scipy}.mat(C)
    Tff= {\bf scipy}.mat(T)
    nodelen = len(Number)
    i = 0
    Num = 0
    while i < nodelen:
        TList = []
        CList = []
        j = 0
        while j < nodelen:
            #Ftt = 'T' + str(Num+i-1) + '_' + str(Num+j-1)
            Ftt = T[Number[i]-1][Number[j]-1]
            TList.append(Ftt)
            #Fcc = 'C' + str(Num+i-1) + '_' + str(Num+j-1)
            Fcc = C[Number[i]-1][Number[j]-1]
            CList.append(Fcc)
            j = j + 1
        i = i + 1
        NewTMatrix.append(TList)
        NewCMatrix.append(CList)
    Tff = {\bf scipy}.mat(NewTMatrix)
    Cff = {\bf scipy}.mat(NewCMatrix)
    TffI = Tff.I
    AMat = TffI*Cff
    lam = {\bf scipy}.linalg.eigvals(AMat)
    return lam

#################################
# TESolve(C, T, FreeNodes, PerscribedNodes)
# Description: This function takes a global T and C matrix and
#   along with a list of FreeNodes and the List of PerscribedNodes(boundaries)
#   solves the needed matrix for the TE mode case.
# Output: lambda values, lambda = k^2
#
# Status: Broken
#################################
def TESolve(C, T, Number, PNodes):
    NewCMatrix = []
    NewTMatrix = []
    for x in PNodes:
        Number.append(x)
    Number.sort()
    Cff = {\bf scipy}.mat(C)
```

```
#print Cff
Tff= {\bf scipy}.mat(T)
#print Tff
nodelen = len(Number)
i = 0
Num = 0
while i < nodelen:
    TList = []
    CList = []
    j = 0
    while j < nodelen:
        #Ftt = 'T' + str(Num+i-1) + '_' + str(Num+j-1)
        Ftt = T[Number[i]-1][Number[j]-1]
        TList.append(Ftt)
        #Fcc = 'C' + str(Num+i-1) + '_' + str(Num+j-1)
        Fcc = C[Number[i]-1][Number[j]-1]
        CList.append(Fcc)
        j = j + 1
    i = i + 1
    NewTMatrix.append(TList)
    NewCMatrix.append(CList)
Tff = {\bf scipy}.mat(NewTMatrix)
Cff = {\bf scipy}.mat(NewCMatrix)
#print Cff
TffI = Tff.I
AMat = TffI*Cff
lam = {\bf scipy}.linalg.eigvals(AMat)
return lam
```

B Program Interface

Below is the program used to access the **pyfem** class. This file was named pyfemtest.py and completely outlines the functions of the **pyfem** class and its typical uses.

```
#!/usr/local/bin/python
########################
#  filename: pyfemtest.py
#  written by: Jason Sidabras
#  Description: This file outlines the use of
#    the functions in the pyfem class
#  Requirements: Python 2.4
#           gmsh 1.63: msh version 1 output
#           {\bf scipy} 0.9.6
#  Usage: (uncomment predetermined filename)
#       $ ./pyfemtest.py
#   or
#       $ ./pyfemtest.py meshfile.msh
########################
import pyfem
import {\bf scipy}
import math
filename = None
# L Shaped
#filename = './Lwaveguide.msh'
# 1 free node
#filename = './SquareWG_1Nodes.msh'
#2 free nodes
#filename = './SquareWG_2Nodes.msh'
# 3 free nodes
#filename = './SquareWG_3Nodes.msh'
# 4 free nodes
#filename = './SquareWG_4Nodes.msh'
# 25 free nodes
#filename = './SquareWG_25Nodes.msh'
# 419 free nodes
#filename = './SquareWG_GMSH.msh'
# Ridged Waveguide
#filename = './RidgeWG.msh'
# Retangle 2/1 WG
#filename = './Rectangle.msh'
# Elliptical Meshed
#filename = './Ellipse.msh'

# If no file is selected from above, use command line argument
if filename == None:
    import sys
    filename = sys.argv[1]

# Get a list of nodes from the file
node = pyfem.nodes(filename)
```

```
# Get a list of elements:
# 1: Line Elements (boundary)
# 2: Triangle Element
trielm = pyfem.Elms(filename, 2)

# For every node number assign x,y coords
trielmRec = pyfem.elmRecurse(trielm, node)

# Create array of element projection matrix
elmCoeff = pyfem.elementCoeff(trielmRec)

# Create global matrix from projection matrix
C = pyfem.GlobalMatrix(elmCoeff, filename)

# Create global T matrix using triangle node list
T = pyfem.TMatrix(trielmRec, filename)

# Get a list of the free nodes
FreeNodes = pyfem.FreeNode(elmCoeff,trielmRec, filename)

# Solve using Global C, T Matrix and FreeNodes
lam = pyfem.TMSolve(C, T, FreeNodes)

# Sort and Print
lam.sort()
print 'TM Modes:'
for x in lam:
    # x is k^2... find k and print it
    print math.sqrt(x)

# Print out TE Modes... if working.
#lam = pyfem.TESolve(C, T, FreeNodes, PNodes)
#lam.sort()
#PNodes = []
#for x in pyfem.Elms(filename, 1):
#    PNodes.append(x[0])
#print 'TE Modes:'
#for x in lam:
#    if x.imag == 0 and x.real > 0:
#        print math.sqrt(x.real)
```

C Example msh file

Below is the example mesh file for a 9 node square matrix with area, 2 x 2.

```
$NOD
9
1 1 1 0
2 0 1 0
3 -1 1 0
4 -1 0 0
5 0 0 0
6 1 0 0
7 1 -1 0
8 0 -1 0
9 -1 -1 0
$ENDNOD
$ELM
16
1 1 1 1 2 1 2
2 1 1 1 2 2 3
3 1 2 2 2 3 4
4 1 2 2 2 4 9
5 1 2 2 2 9 8
6 1 2 2 2 8 7
7 1 2 2 2 7 6
8 1 2 2 2 6 1
9  2 100 100 3 6 1 5
10  2 100 100 3 5 1 2
11  2 100 100 3 5 2 3
12  2 100 100 3 4 5 3
13  2 100 100 3 9 5 4
14  2 100 100 3 9 8 5
15  2 100 100 3 8 7 5
16  2 100 100 3 5 7 6
$ENDELM
```

D Global Matrix

E PDE to Functional