A new and simple technique for vectorization of Finite Element Method in electromagnetics

C. H. Veloso 1, A. M. F. Frasson 2 and K. Z. Nobrega *

1, DEE-Instituto Federal do Maranhão/ 2 DEE-Universidade Federal do Espírito Santo

1 Av. Getúlio Vargas 4, Monte Castelo 65030-005 (São Luís-Ma, Brazil), 2 Av. Fernando Ferrari 514, Goiabeiras 29075-910 (Vitória-Es, Brazil)

1 henri_5_veloso@hotmail.com; 2 antonio@denise.ele.ufes.br; * bzuza@ifma.edu.br

Abstract— Here it will be present a new approach on programming the finite element method that turns programs faster, simpler and more effective in languages like MATLAB or SCILAB, for example, when compared to the traditional way to program and that uses built-in function. Through this new approach, Laplace’s Equation in a 2D domain has been solved getting a better performance up to 2.5 for a linear mesh with triangles and more than 1,000,000 nodes. Finally, some remarks will also be done about other possible uses of the technique.

Index Terms—Finite element methods; Laplace equation; MATLAB.

I. INTRODUCTION

During the last twenty years, MATLAB has been widely used as a potential tool for scientific programming, especially for solving of differential equation either in the industry or in the classroom [1]-[6]. Once it is an interpreted language, MATLAB is usually slower than compiled languages like C or FORTRAN although it won’t be true for all kind of application. On reality, Matlab can be well adapted to numerical computation since the underlying algorithms for built-in functions and supplied m-files are based on the standard libraries LINPACK and EISPACK, usually compiled in C.

Here, it will be reported an effective, simple, vectorized and easily adaptive way to program the finite element method (FEM) in MATLAB using built-in functions only (except for one) with no loops. In fact, spent times of simulation comparing a code with loop and the one proposed will be shown. The case of study was Laplace’s equation in a 2D domain with nodal formulation. Although not described here due to page limitations, a simple approach to generate the element matrices applied into the proposed code is also discussed and it will be explained on the next section.

II. NODAL FORMULATION FOR 2D DOMAIN

As it is well-known, the FEM formulation with triangular elements can be described in four simple steps:

1. in the local coordinates \((\xi, \eta)\), the shape functions \(N_i\) are written as

   \[
   \begin{bmatrix}
   N_1 \\
   N_2 \\
   N_3 \\
   N_4 \\
   N_5 \\
   N_6 \\
   \end{bmatrix} = \frac{\begin{bmatrix}
   L_1 \\
   L_2 \\
   L_3 \\
   L_4 \\
   L_5 \\
   L_6 \\
   \end{bmatrix}}{2L_1 - 0.5} \text{ or } \frac{\begin{bmatrix}
   L_1 \\
   L_2 \\
   L_3 \\
   L_4 \\
   L_5 \\
   L_6 \\
   \end{bmatrix}}{2L_2 - 0.5}
   \]

   \[
   \frac{\begin{bmatrix}
   L_1 \\
   L_2 \\
   L_3 \\
   L_4 \\
   L_5 \\
   L_6 \\
   \end{bmatrix}}{4L_1L_2} \text{ or } \frac{\begin{bmatrix}
   L_1 \\
   L_2 \\
   L_3 \\
   L_4 \\
   L_5 \\
   L_6 \\
   \end{bmatrix}}{4L_3L_4}
   \]

   \[L_1 = 1 - \xi - \eta \quad L_2 = \xi \quad L_3 = \eta \quad L_4 = L_5 = L_6 = \frac{1}{2} (L_1 + L_2 + L_3)
   \]

   with \(L_4 = L_5 = L_6\)

2. The coordinates \(x, y\) become:

   \[
   \begin{align*}
   x &= x_i L_1 + x_j L_2 + x_k L_3 \\
   y &= y_i L_1 + y_j L_2 + y_k L_3
   \end{align*}
   \]

   \[
   \begin{align*}
   x &= x_i L_1 + y_k L_3 + y_i L_k \\
   y &= x_j L_2 + y_j L_k + y_i L_k
   \end{align*}
   \]

   \[
   \begin{align*}
   x &= x_i L_1 + y_k L_3 + y_i L_k \\
   y &= x_j L_2 + y_j L_k + y_i L_k
   \end{align*}
   \]

3. An integration in global coordinates can be transformed in local coordinates as:

   \[
   \int_{\Delta} f(x,y)dxdy = \int_{\Delta} f(\xi,\eta)J(\xi,\eta)|d\xi|d\eta
   \]

   \[
   \int_{\Delta} f(x,y)dxdy = \int_{\Delta} f(\xi,\eta)|d\xi|d\eta
   \]

   \[
   \int_{\Delta} f(x,y)dxdy = \int_{\Delta} f(\xi,\eta)|d\xi|d\eta
   \]

4. Finally, the last step concerns differential operations, it means:

   \[
   \frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}
   \]

   \[
   \frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial \eta}
   \]

   \[
   \frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}
   \]

   \[
   \frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial \eta}
   \]

   \[
   \frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}
   \]

   \[
   \frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial \eta}
   \]

The idea to program these four steps is to easily and automatically produce the element matrices to be introduced into the main program discussed in next section. In fact, the relations expressed by Eqs (1) to (5) already contemplate the linear or quadratic elements and they can be easily modified to 3D domain.

III. CONCEPT OF THE VECTORIZED CODE

At first, it is assumed that mesh data have already been loaded. For the Laplace’s problem those data are saved on three variables: nodes, topol and bound. nodes is a matrix containing coordinates of all points of discretization and its dimension will be \(n_x \times n_y\). The variable topol contains mesh’s topology and it is a matrix \(n_x \times ie\) (ie=3,6 for linear or quadratic elements, respectively). Finally, bound has dimension \(n_x \times 2\) where the first and the second columns correspond to number of the node and the value associated to it, respectively. It will be used to assign tension’s value in the already known nodes. Following, one has the possible code for implementation, explained line by line.

As someone knows, MATLAB has to find and to assign a contiguous piece of free RAM memory for a given variable. For this reason, the following coordinates \(x_i\) and \(y_i\) have not used into the same variable but for different ones. Actually, they demand the same RAM memory size but in the process of allocation it is faster writing in this way because different variables are smaller if compared to a single one. Thus, until
the line of $A_e$ one will have the area of all elements that is based on $x_i$ and $y_i$ only.

Next, the line code has intention to create two vectors $col$ and $lin$ associated to a matrix $ie 	imes ie$. The idea is to create two vectors containing the position of a matrix $ie 	imes ie$ and use the indices to pipe all local nodes from $topol$. For example, from the following line code produces $lin=[1 1 2 2 3 3]$ and $col=[1 2 3 1 2 3]$ where $lin$ makes the rule of changing the weight function in the FEM while $col$ the tension (unknowns).

It must be emphasized that ind2sub or any equivalence is the only non built-in function along this whole code, it means, the only command line slower if compared to C or FORTRAN but its idea is too simple and truly fast even for 3D domain.

After that, all those combinations (for all elements at the same time) will be stored into variable $biunivoc$ which keeps the position where the element matrices (here assembled for all elements) must be pre-allocated into the global matrix.

Then, taking the expression for the element matrix generated by any routine following steps of section 2, one assign $biunivoc$ as the pseudo global matrix to finally assemble the global matrix $A$, through the sparse function.

$$A_{aux} = \begin{bmatrix}
(x2 - x3)/4/Ae & (y2 - y3)^2/4/Ae & 0 & \cdots & 0 \\
(y2 - y3)^2/4/Ae & (x3 - x2)/4/Ae & 0 & \cdots & 0 \\
0 & 0 & (x2 - x3)/4/Ae & (y2 - y3)^2/4/Ae & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & \cdots & (x1 - x2)/4/Ae
\end{bmatrix}$$

Finally, the last command lines are self-explained.

According to Table 1, one can see how the vectorized code is more efficient than the conventional one for a factor about 2.5 even for a number of nodes above 1M. Also, one should remember that only built-in function have been used, which turns the code with a processing similar to language C. Although not shown, similar results are also obtained with quadratic elements.

Above all, notice that for quadratic elements the only line code that changes is the element matrix $A_{aux}$, turning this approach very simple. Besides, for the best of our knowledge all vectorized codes still have a loop, which is completely absent in this one. As a last remark, this idea can be easily extended for other kinds of problems like mode analysis or beam propagation no matter dimension of the mesh and even considering conditions like PML, for example.

Finally, authors would like to thank you both FAPEMA and IFMA for the financial support of this research project.

### REFERENCES