Computational Techniques to Solve Electromagnetic Problems by Using FEM

N.FÜSUN SERTELLER [(*)] , DURSUN ÜSTÜNDAĞ[(**)]
* Electrical Education Department, Faculty of Technical Education,
**Mathematics Department, Faculty of Art and Science
Marmara University, Ziverbey Campus 34722, Istanbul/TURKEY.
e-mail: fserteller@marmara.edu.tr, dustundag@marmara.edu.tr

Abstract: Numerical modeling is nowadays an essential tool for all researchers involved with the field of electrical, mechanical engineering, physics and etc. The object of this paper is to provide a clear, easy and understandable study on FEM (finite element method) using MATLAB and MATHEMATICA, which both provide powerful support to solve large equations. 1D and 2D FEM solutions are studied on electromagnetic (EM) field distribution which is one of the essential parts of electrical engineering. The results from the two different programs were compared with those obtained by using the finite difference method (FD). Computer programs and formulations of the finite element technique have been constructed to make the concept more understandable and explicit for engineering students and practicing engineers.

Key-Words: MATLAB, Mathematica, FEM, FD, Electrostatic Analysis.

1 Introduction

In the past few decades, numerical analysis methods have come to prominence and became more appealing with the advent of fast digital computers. In the study of electrostatics the most commonly used numerical techniques for the numerical solution of the partial differential equations is finite difference (FD) and finite element method (FEM). Both methods[22] are based on some discretization of the fields into a collection of the points or cells so that the differential equations are approximated by a set of algebraic equations on this collection. This system of algebraic equations is then solved to produce a set of discrete values which approximates the solution of the differential systems over the fields. Furthermore, FD uses point-wise approximations of the governing equations, while FEM uses piecewise or regional approximations. Although numerical methods give approximate solutions, these solutions are sufficiently accurate for electrical engineering purposes[27,28]. Without computers, solving the numerical equations is very complicated and time consuming. An intersection of Computational science and electrical engineering (CSEE) is a new and rapidly evolving field. O'Leary defined CSEE as "an interdisciplinary approach to the solution of the problems in the natural science and technology, drawing on the tools of a science or engineering discipline plus computer science plus mathematics". This definition of CSEE can simply be illustrated in Figure 1. Furthermore, as the software enables visualization, the term CSEE can also be extended to encompass computational and visual electromagnetic for electrical engineers (CVEMEE) [9].

In this study, using MATLAB and MATHEMATICA programming languages some computer programs were developed for finding solution of Poisson type equations, which are commonly used to define the magnetic field of the

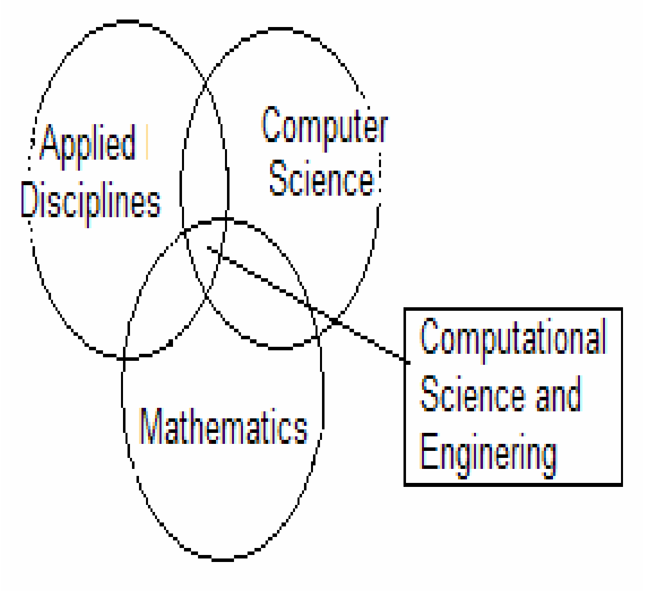

Fig.1 The definition of computational science and engineering

electrical systems. It is assumed that the internal potential values within any domain are initially empty. In the EM problems, the region is chosen as the most popular and the most explicit state. This is because of the complexity of the numerical methods. This kind of structure is seen generally in transmission lines, earthing systems and in electrodes with a gap between the electrodes.

2 Comparisons of Programming Features

This study assesses the similarities between two powerful software programs, namely MATLAB[7] and MATHEMATICA[21], which are important for engineers and scientists. MATLAB started as an interactive program for doing matrix calculations and has now grown to a high level mathematical language that can solve integrals and differential equations numerically and plot a wide variety of two and three dimensional graphs. On the other hand, MATHEMATICA is a state-of-the-art and powerful system for doing mathematics by computer. It has steadily grown in breadth and depth to become today an unparalleled platform for all forms of computation. Although MATLAB is widely-used among engineers, it is more suitable for numerical solutions of engineering problems. MATHEMATICA is designed for doing symbolic computations rather than numerical ones. MATHEMATICA has inbuilt converters for converting to or from other languages and file formats such as FORTRAN and C. This means that the users of these languages are able to adapt easily their programs to MATHEMATICA.

In this study, it is shown how MATLAB and MATHEMATICA are suited for engineering computations. Let us see some examples:
The first one is to calculate the element coefficient matrix and the potential values of an area in MATLAB and MATHEMATICA respectively. In MATLAB,

```
ce=(p*p'+q*q')/(4.0* Area)
v=Inv(ce)*b;
```

and in MATHEMATICA,

```
ce=Outer[Times, p, p]+Outer[Times,q,q]/(4.0 Area)
v=Inverse[ce].b
```

where “p” and “q” are local coordinate vectors, “ce” is a global coefficient matrix and “b” the right hand vector. MATHEMATICA does not understand the transpose of one dimensional matrix, but the symbol (’) implies the transpose of the vector in MATLAB automatically [1, 2, 3]. Although a vector “q” can be multiplied by its transpose using (’) in MATLAB, this can be done by calling a special function called “Outer” in MATEMATICA. In MATLAB, The symbol (*) is used for an ordinary multiplication and a matrix product operator. However, MATHEMATICA separates this multiplication and uses (*) for an ordinary multiplication and the symbol (.) for the matrix product.

Standard MATLAB is not capable of telling you that the cosine function is the derivative of the sine function. So when we plotted the sine function in MATLAB we have done all the hard work implicitly. With the command 'x=0:0.01:2*pi' we have selected the sample points and with 'plot(x, sin(x))' we have told MATLAB to plot the sample points and use an interpolation technique to find (and plot) the function values in between the samples. Now compare this with the way MATHEMATICA operates. The command to plot the sine function would look something like:

```
Plot[Sin[x], {x, 0, 2 Pi} ]
```

Now it is MATHEMATICA who decides on the sampling interval (in fact it is not even a uniform sampling, i.e. more samples are taken where needed to obtain an accurate plot).
As a programming language, MATHEMATICA and MATLAB contain programming structures that are similar to those in other programming languages such as function definition, looping structures ect. But they use quite different syntax and punctuation conventions for coding. However MATHEMATICA allows us to program in several different styles, including procedural, rule-based, mathematical, and functional programming, mirroring the styles of programming used in BASIC, FORTRAN, C, Lisp, APL, and many other languages. As a result, MATHEMATICA and MATLAB provide “grey boxes” that is of sufficiently high level to avoid unnecessary details, yet visible enough to be modified and customized by users.

3 Governing Equations

This work considers Poisson type electromagnetic equation in a region whose two-dimensional profile is given in Fig. 2:

$$\nabla^2 V = -\frac{\rho_s}{\varepsilon}, \tag{1}$$

where ρ_s charge density, ε permittivity of medium. For 2D, one horizontal outside boundary value of $50V$ and a vertical boundary of $30V$ were used. Inside the boundary, the value is zero. We consider only one quarter part of the device $(x, y \geq 0)$. If the solution region is charge free $\rho_s = 0$, the Equation (1) turns into Laplace's equation:

$$\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \qquad (2)$$

Given the size $(x_n \times y_n)$ of the solution region in Fig.2, the goal is here to divide the region into subregions. Suppose that n_x and n_y are numbers of divisions in x and y direction. The total number of elements ne, nodes n_d and the boundary n_p are then given:

$$\begin{aligned} ne &= 2n_x n_y \\ n_d &= (n_x+1)(n_y+1). \\ n_p &= 2(n_x+n_y) \end{aligned} \qquad (3)$$

On the solution region, a systematic way of numbering the elements and nodes are completed. It is easy to model uniform and nonuniform meshes. A mesh is also uniform if all Δx and Δy are equal, otherwise, a nonuniform mesh is preferred if it is known in advance that the parameter of interest varies rapidly within some part of the solution domain. This allows a concentration of relatively small elements in the regions where the parameter changes rapidly and particularly since these regions are often of greatest interest in the solution [10,11,24].

Beside nodes and elements numbering, we should consider bandwidth reduction on FEM analysis. The bandwidth of the global coefficient matrix depends on the nodes numbering. In order to minimize the bandwidth, we must number the nodes across the narrowest part of the region.

The approximate solution for the whole region is

$$V(x,y) \cong \sum_{e=1}^{N} V_e(x,y) \qquad (4)$$

and the energy per unit length associated with the element e is given by the following Equation:

$$W_e = \frac{1}{2}\int \varepsilon |E|^2 ds = \frac{1}{2}\int \varepsilon |\nabla V_e|^2 ds \qquad (5)$$

This can be written in matrix form:

$$W_e = \frac{1}{2}\varepsilon [V_e]^T \left[C^{(e)}\right][V_e] \qquad (6)$$

where T denotes the transpose of the matrix and $\left[C^{(e)}\right]$ is the coefficient matrix. Finite differences are solved using general formulation of Equation (2), which is the three point approximation used for the second derivative. Thus $V(x,y)$ transforms to the following equation [2,3,4]:

$$V_{i,j} = \frac{1}{4}(V_{i+j,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}) \qquad (7)$$

For simplicity, we use the notation that the index (i,j) indicates the coordinate (x_i, y_j).

On the other hand, 1D electromagnetic equation can be given by the following equation:

$$\Phi'' = -k \qquad (8)$$

In order to solve this boundary value problem using finite differences, we obtain

$$\Phi_i = \frac{1}{2}(\Phi_{i+1} + \Phi_{i-1} + kh^2) \qquad (9)$$

where $h = 1/N$. The entire domain is divided N equal segments each of length as in Figure 2 so that there are $N+1$ nodes. By integrating 1D electromagnetic Equation (8) with respect to x we obtain the exact solution[1,18] in the following form:

$$\Phi = -\frac{kx^2}{2} + Ax + B \qquad (10)$$

4 Numerical Results

To implement the methods for solving one and two dimensional Poisson and Laplace type problems shown in Fig.2 - 3, some programs were developed in both MATHEMATICA and MATLAB and are given in Appendix. Although the programs can be used for 1D, 2D and 3D EM problems, their general usages are only given for 1D and 2D. In this paper, we therefore attempt to solve 1D and 2D EM problems[24,26].

The 1D EM problems involve mostly magnetic induction, magnetic flux and magnetic field for electrical engineers and researchers. The solution is done for 1D since only z direction of the Poisson equation (Eq.1) is handled. The first example was run in Mathematica, the second using MATLAB. In this analysis 100 and 300 elements was used with the suitable iteration number in which the numerical solutions should be in a good agreement with exact solution. Poisson type equation (Eqs.8 and 9) was developed for two different kind right hand sides (see in appendix 1D programs).

For 2D, the solution region which has been investigated, given in Figure 3, whose dimensions are in meters. In the FEM analysis, 83 nodes and 112 triangular elements are chosen. This indicates a relatively fine meshing structure for this work. The linear system constructed according to Equation (4) is solved by the preconditioned conjugate gradient [5,6,9,25]. The numerical results are tested by the FD method, based on 37 boundary elements for outside boundaries and 47 nodes inside the region. In both methods, numerical solutions are obtained at the same spatial locations.

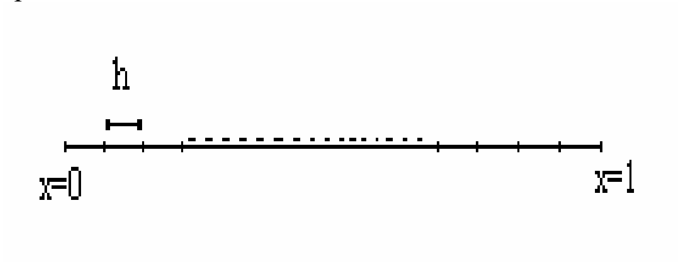

Fig.2 One dimensional example

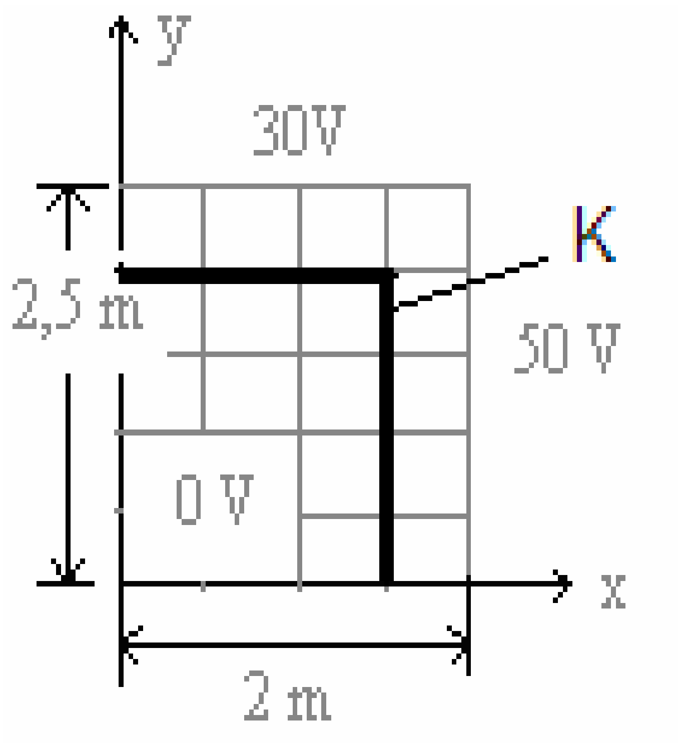

Fig.3 Solution domain for finite elements

To solve the problem using FEM a mesh is applied to Fig. 3 and FD is used in testing FEM's accuracy.

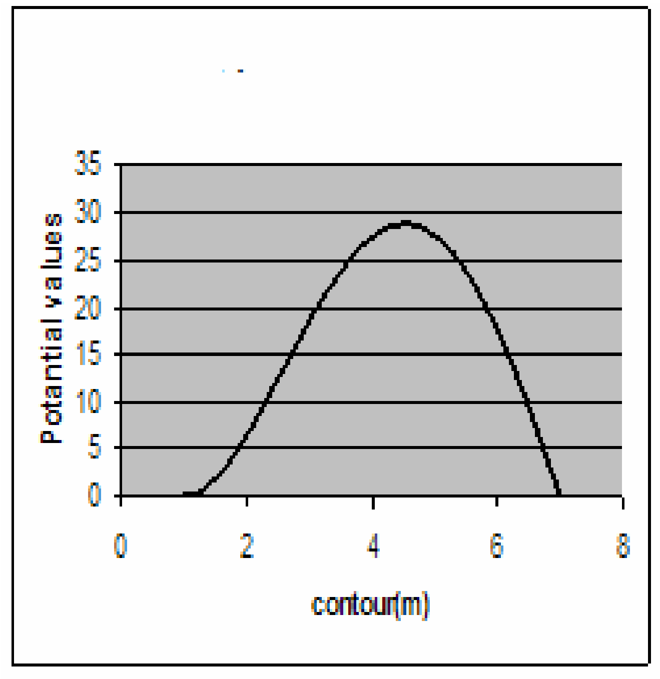

Fig.4. Potentials values along the contour.

This solution was calculated using the MATHEMATICA program and results are shown in the Figs 3, 4 and 5. The FEM and FD programs give equal values at the same spatial nodes.

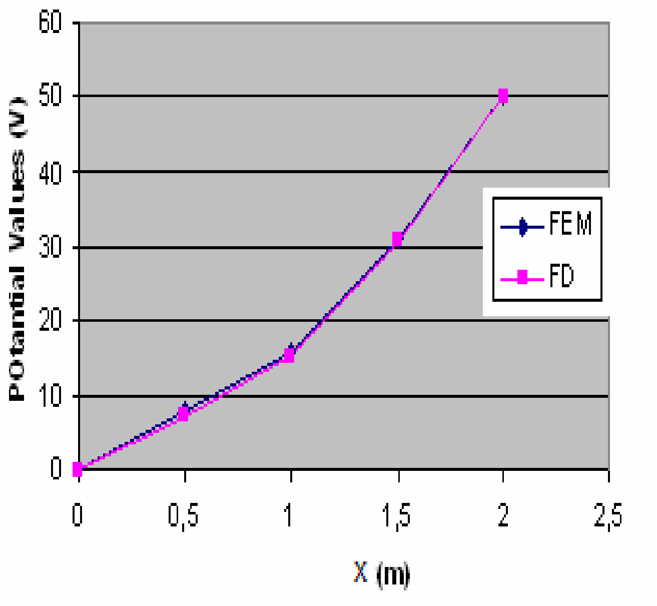

Fig.5. Potential Values versus $x(m)$. Comparison the FEM and FD along x coordinate.

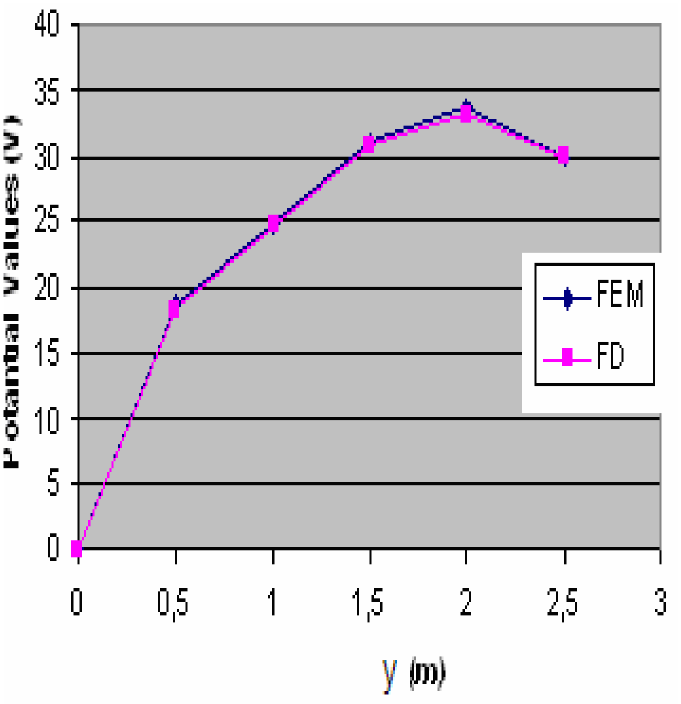

Fig.6. Potential Values versus $y(m)$ FEM and FD comparison along the y coordinate.

Figure 7 illustrates the computer output of the 1D MATHEMATICA program. The exact and the numerical results are also compared on the same graph shown in Figure 8. After developing the programme, the numerical results can easily be visualized.

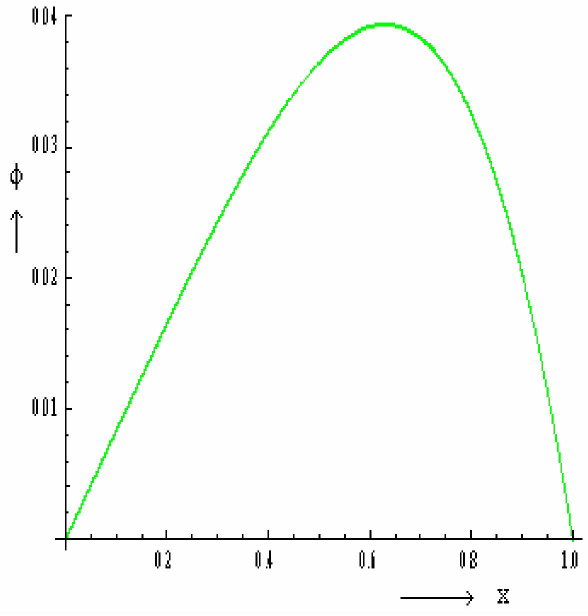

Fig.7 One dimensional FEM analysis results

The FEM analysis graph shown in Fig.7 was given separately, since the exact solution and FEM solution coincide on the same spatial points.

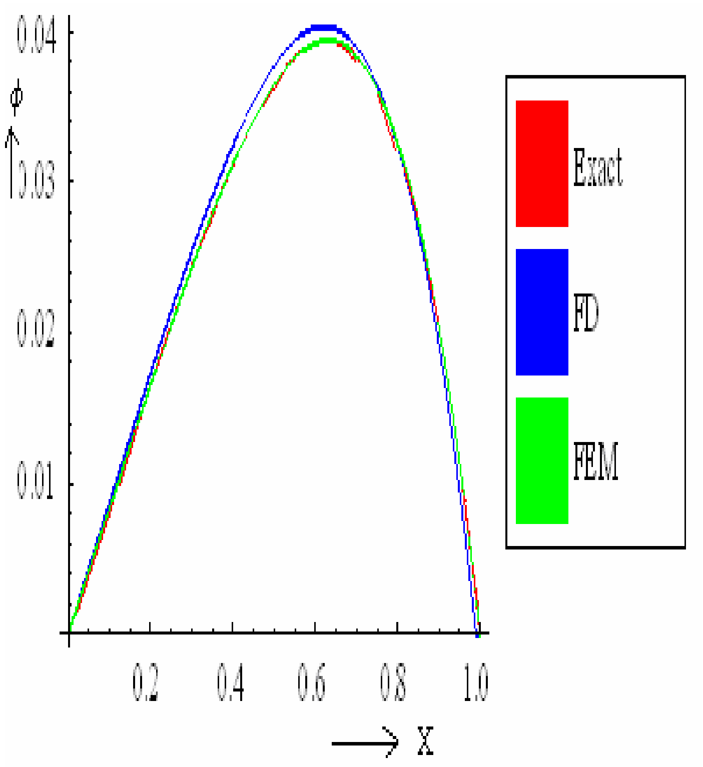

Fig.8. Comparison of 1D EM numerical and exact solution.

The iteration number plays an important role in the numerical solution of the equations because the numerical solution approaches to exact solution as the iteration number increases.

5 Conclusions

Although simulation programs exist that greatly simplify problems relating to EM, and FEM and FD, they are not adequate for engineering design and analyses. Solving a problem by limiting the model within defined boundaries also means reducing the accuracy of the solution. This study was carried out for FEM using MATHEMATICA and MATLAB programs. It was observed that MATHEMATICA can easily be used as a programming language like MATLAB, and has greater functionality than MATLAB in some areas, for instance doing symbolic computations rather than numerical ones. The FEM results were compared with those obtained by using the FD method, coded in MATHEMATICA and MATLAB. Finally, the results of the two software programs were found to be closely matched. The comments presented in this paper can serve as an important introduction to a course of more complex computational methods.

The ability to use higher order languages such as MATHEMATICA and MATLAB is becoming an important skill sought by industrial employers in today's competitive environment. Therefore, the ability to program in low-level computer languages such as C or FORTRAN will become less relevant to industry in the nearest future.

References:

[1] M.N.O. Sadiku, *Elements of Electromagnetic,* Oxford university press, 2001.

[2] O.C. Zienkiewicz, R.L. Taylor, *The Finite Element Method*, McGraw Hill Company, 1989.

[3] E. Don, *Mathematica Schaum' Outlines*, Mc Graw Hill Company, 2001.

[4] B. Klaus-Jürgen, *Finite Element Method Procedures*, prentice Hall, 1996.

[5] D.P. O'Leary, Computational science and applied mathematics, *IEEE Computational Science & Engineering,* pp.13-18, 1997.

[6] J. Ju, D.V. Thiel, Computational and visual Electromagnetic using an Integrated Programming language for Undergraduate Engineering Students, *IEEE transaction on magnetic*, Vol.36, No.4, pp.1000-1003, 2000.

[7] B. M. Brenier, *MATLAB for Engineers*, Addison-Wesley, 1995.

[8] M.K. Haldar, Introducing the Finite Element Method in Electromagnetic to Undergraduate Using MATLAB, *International Journal of Electrical Engineering Education*, Vol. 43, pp. 232-244, 2006.

[9] L.C. Agba, M. Sadıku, A. Makki, A Further Introduction to Finite Element Analysis of Electromagnetic Problems, *IEEE transaction on Education*, Vol. 34, No. 2, pp. 322-329, 1991.

[10] J. Ju, D.V. Thiel, Numerical Techniques in Electromagnetic and Communications- A PC Based Third Year Undergraduate Subject for Microelectronic Engineering, *IEEE AP-S International Symposium and URSI Radio Science meeting*, USA, pp. 111-114, 1994.

[11] E. Hinton, D.R.J. Owen, *An introduction to finite element calculation,* Swansea, U.K. Pineridge, pp. 247-260, 1980.

[12] T. Akhlanghi, Finite Element Lower Bound Limit Analysis in Soil Mechanics Using Nonlinear Programming, *Wseas Transaction on Advances in Engineering Education,* Vol. 3, No. 8, 2006.

[13] K. LAPIN, S. RAGAISIS, Integrating team projects into the SE Curriculum, *Wseas transaction on Advances in Engineering Education,* Issue 3, Volume 5, March, 2008.

[14] J.C.M., KAMPE, *Division of Engineering Fundamentals, MATLAB Programming*, Virginia Polytechnic & State University, 1999.

[15] E. Hall, Mathematical and Visualization Software, ITC Research Computing Support Group, www.itc.virgina.edu.tr.

[16] V.L. Charles, *Introduction to Scientific Computing: A Matrix Vector Approach Using MATLAB*, Printice Hall, 1997.

[17] O. Preis, T. Biro, I. Ebner, I. Ticar, An Electromagnetic Field Analysis Tool in Education, *IEEE Transaction on Magnetics*, Vol. 38, No.2, pp.1317-1320, 2002.

[18] M. Eduardo, D. Tobiac, Experimental Study of the Neumann and Dirichlet Boundary Conditions in Two-Dimensional Electrostatic Problems, *American Journal of Physics*, pp. 70 -12, 2002.

[19] K.F. Warnick, R.H. Selfridge, D.V. Arnold, Teaching Electromagnetic Field Theory using Differential Forms, *IEEE Transaction on Education*, Vol. 46, No.1, pp. 53-68, 1997.

[20] K.R. Richter, K. Preis, H. Stoegner, Numerical field Calculations in Electrical Engineering Education, *IEEE MELECON'83*, Volume1, Athens, Greece.

[21] P. Wellim, R. Gaylord, S. Kamin, *An Introduction to Programming with Mathematica*, Cambridge University Press, 2005.

[22] G.D. Smith, *Numerical Solution of Partial Differential Equations :Finite Difference Methods*, Oxford University Press, 2004.

[23] M. Rizzi, M. Maurantonio, B., Castagnolo, 3d Finite Element Model for GaAs α – particles pixel detector, *WSEAS Transaction on Electronic,* Issue 4, Volume 1, October, 2004.

[24] K.Aniserowicz, Comparison of Diffrent Numerical Methods for Solving Boundary-Value Problems in ElectroMagnetics, *IEEE Transactions on Education*, Vol. 47, No.2, pp. 241-246, 2004.

[25] M.N.O. Sadiku, A Comparison of Numerical Methods for Computing Electromagnetics Fields, *Conference Proceedings IEEE Southeastcon*, Vol. 1, pp. 42-47, 1990.

[26] P.P. Silvester, R.F. Ferrari, Finite Element for Electrical Engineer, Cambridge Univerisity Press, 1983.

[27] O.C. Zienkiewicz, K. Morgan, Finite Element and Approximation, Nw York: Wiley-Interscience, 1982.

[28] J.L. Volakis, A. Chatterjee, L.C. Kempel, Finit Element Method for Electromagnetics, Piscataway, NJ: IEEE Press, 1998.

Appendix

(* Finite Difference Method for 2D*)

```
 (* v1:  Vertical Potential *)
(* v2:  Horizontal Potential*)
(* v3 :  Potential inside the region*)
(* ni:  iteration number*)
(* nx:  node number of x coordinate*)
(*ny:  node number of y coordinate*)
(* v:   Potential values inside the region*)
Clear[ni,nx,ny];
v1=30.0;
v2=20.0;
v3=0.0;
ni=200.0;
nx=9.0;
ny=11.0;
(* Input File *)
v=Table[0,{nx},{ny}]
For[j=1,j≤ny-1,j++, v[[1,j]]=v3;
   v[[nx,j]]=v1;
   ];

For[i=1,i≤nx-1,i++,
   v[[i,ny]]=v2;   v[[i,1]]=v3;
   ];
v[[nx,1]]=15.0;
v[[1,ny]]=10.0;
v[[nx,ny]]=25.0;
For[k=1,k≤ni,k++,
   For[i=2,i≤nx-1,i++,
      For[j=2,j≤ny-1,j++,
If[i≤5∧j≤5,v[[i,j]]=v3;,
           v[[i,j]]=0.25  (v[[i+1,j]]+v[[i-
1,j]]+v[[i,j+1]]+v[[i,j-1]]);]
        ];
      ];
   ];
m1=Do[Print[v[[i,j]]],{i,2,4},{j,2,5}]
Do[Print[v[[i,j]]],         {i,1,nx},{j,1,ny}]
Print["v1=",v[[2,5]],"------"        "v2-=",v[[2,4]],"------
""v3-=",v[[3,5]],"-------""v4=",v[[3,4]],"-------","v7 =-
",v[[4,3]]]
```

and

Finite Difference 2D [Matlab]

```
Clear[ni,nx,ny];
v1=30.0;
v2=20.0;
v3=0.0;
ni=200.0;
nx=9.0;
ny=11.0;
% Input File
v=zeros(nx,ny);
For j=1:ny-1
   V(1,j)=v3;
   v(nx,j)=v1;
   end

For i=1,i≤nx-1,i++,
   v(i,ny)=v2;
   v(i,1)=v3;
   end
v(nx,1)=15.0;
v(1,ny)=10.0;
v(nx,ny)=25.0;

For k=1:ni
   For i=2:nx-1
      For j=2:ny-1
         If(i≤5 and j≤5,v(i,j)=v3;
           v(i,j)=0.25 *(v(i+1,j)+v(i-1,j)+v(i,j+1)+v(i,j-
1));
            end
      end
   end

diary a:test1:out
[v(2,5) , v(2,4), v(3,5),v(3,4),v(4,3)]
[[1:nx,1:ny]  v(i,j)]
diary off
```

(* Finite Element Method for 2D*)

```
(* nd:  node number*)
(*c:   global coefficient matrix*)
(*ce: element coefficient matrix *)
(*np:  fixed element number*)
(*val:  boundary element value*)
(*b:  right hand side matrix*)
(*ndp:  boundary node number*)
Clear[nd,c,ce,np,nd,ne,b,nl,x,y,val,ndp];
nd=83;
ne=112;
(* Input File *)
{n1,np,x,y,pot,npot}
p=Table[0,{3}];
c=Table[0,{nd},{nd}];
b=Table[0,{nd}];
For[i=1,i ≤ ne,i++,
  knodes=nl[[i]];
  xl=x[[knodes]];
  yl=y[[knodes]];
  Print[yl,xl];
  p=Table[0,{3}];
  q=Table[0,{3}];
  p[[1]]=yl[[2]]-yl[[3]];
  p[[2]]=yl[[3]]-yl[[1]];
  p[[3]]=yl[[1]]-yl[[2]];
  q[[1]]=xl[[3]]-xl[[2]];
  q[[2]]=xl[[1]]-xl[[3]];
  q[[3]]=xl[[2]]-xl[[1]];
```

```
 alan=0.5 *Abs[p[[2]]* q[[3]]-q[[2]]* p[[3]]];

 (* Determine coefficient Matrix *)
  ce=(Outer[Times,p,p]+Outer[Times,q,q])/(4.0  alan);
  Print[MatrixForm[ce]];

 (* Assemble *)
  For[j=1,j≤3,j++,
   ir=nl[[i,j]];
   Print[ir];
   iflag1=0;
   For[k=1,k≤np,k++,
    Print["k=",k];
    If[ir==ndp[[k]],
     c[[ir,ir]]=1;
     b[[ir]]=val[[k]];
     iflag1=1;
     ];
    ];
   If[iflag1==0,
    For[i=1,i≤3,l++,
     ic=nl[[i,l]];
     iflag2=0;
     For[k=1,k≤np,k++,
      If[ic==ndp[[k]],
        b[[ir]]-=ce[[j,l]]* val[[k]];
        iflag2=1;
        ];
      ];
     If[iflag2==0,
      c[[ir,ic]]+=ce[[j,l]];
      ];
     ];
    ];
   ];
  ];
v=Inverse[c].b;
Print["Node","    x    ","    y   ","     v    "];
Print["--------------------------------------"];
Do[Print[i,"       ",x[[i]],"       ",y[[i]],"
",v[[i]]],{i,1,nd}];
```

(* Finite Difference 1D Poisson Type Problem with constant k (k=1.5)*)

```
(*h mesh size*)
(*ni=no. of iteration desired*)
(*n=no. of elements*)
Needs["PlotLegends`"]
Clear[x,x1,h,phi,phiExact];
n=100;
ni=10000;
l=1.0;
h=N[1/n];
x=h Range[0,n];
x1=Drop[x,{1,2}];
phi=Table[0,{n+1}];
For[k=1,k<=ni, k++,
For[i=2,i<=n-1, i++,
phi[[i]]=(phi[[i+1]]+phi[[i-1]]+  1.5 h^2)/2;];];
(*calculate exact value*)
constant=0.75;
phiExact[x0_]:=constant x0 (1-x0);
g2=ListPlot[Transpose[{x,phi}],Joined->True,PlotStyle→{{Blue,Thickness[.005]},{Green,Thickness[.005],Dashing[{.02}]}},DisplayFunction→Identity];
g3=Plot[phiExact[x0],{x0,0,1},PlotStyle→{{Red,Thickness[.005]},{Green,Thickness[.005],Dashing[{.02}]}},DisplayFunction→Identity];
ShowLegend[Show[g3,g2,g1,PlotRange->All,DisplayFunction→$DisplayFunction],{{{Red,"Exact"},{Blue,"FD"},{Green,"FEM"}},LegendPosition->{1.5,-0.5}}]
```

Following is MATLAB software:

% 1D Possion Equation k=1.5

```
% Finite difference method
% Matlab sofware
n=100;
ni=1000;
l=1.0
h=1/n
phi=zeros(n+1,1);
for k=1:ni
phi([2:n])=[phi(3:n+1)+phi(1:n-1)+1.5 *h^2]/2;
end
%Exact value calculation
Phiex=075*x (1.0-x)
Diary a:test.out
[[1:n+1]'phi phiex]
Diary off
```

[14,15,16]

The following 1D Poisson type equation by using Finite element method:

(* 1D Finite Element *)

```
Clear[nd,c,ce,np,nd,ne,b,nl,x,val,ndp];
nd=101;
np=2;
ne=nd-1;
(* Input File *)
nl=Table [{i, i+1},{i,1,nd-1}];
delta=1/ne;
x=Table [(i-1) delta, {i, 1, nd}];
val= {0.0, 0.0}
ndp= {1, nd};
c=Table [0, {nd}, {nd}];
b=Table [0, {nd}];
```

```
Qvec=Table [0, {nd}];
p=Table [0, {2}];
For [i=1,i□ne,i++,
   knodes=nl [[i]];
   xl=x[[knodes]];
   p[[1]]=1.0;
   p[[2]]=-1.0;
   l=xl[[2]]-xl[[1]];
   Qvec[[i]]=1.5 l;
      (* Determine coefficient Matrix *)
  ce0=Outer[Times,p,p];
  ce=ce0/l;
   b=Qvec;

   (* Assemble *)
   For [j=1, j≤2, j++,
     ir=nl [[i, j]];
     iflag1=0;
     For [k=1,k≤np,k++,
       If [ir= =ndp[[k]],
          c[[ir,ir]]=1;
          b[[ir]]=val[[k]];
          iflag1=1;
          ];
       ];
     If[iflag1= =0,
       For [l=1,l≤2,l++,
          ic=nl[[i,l]];
          iflag2=0;
          For[k=1,k≤np,k++,
            If[ic= =ndp[[k]],
               b[[ir]]-=ce[[j,l]] val[[k]];
               iflag2=1;
               ];
            ];
          If[iflag2= =0,
            c[[ir,ic]]+=ce[[j,l]]
            ];
          ];
       ];
     ];
   b[[i]]+=Qvec[[i]];
   ];
   v=Inverse[c]. b;
   Print["Node","     x     ","       v     "];
   Print ["---------------------------------------"];
   Do [Print [i,"        ", x [[i]],"        ", v [[i]]],{i,1,nd}];
   g1=ListPlot[Transpose[{x,v}],Joined-
>True,PlotStyle→{{Green,Thickness[.005]},{Green,T
hickness[.005],Dashing[{.02}]}},DisplayFunction→Id
entity]
```

(*One dimensional Problem Poisson type right hand side x^2*)
(*h mesh size*)

```
(*ni=no of iteration desired*)
Needs["PlotLegends`"]
Clear[h,phi];

n=300;
ni=9000;
l=1.0;
h=N[1/n]
phi=Table[0,{n+1}];
x=h Range[0,n];
x1=Drop[x,{1,2}]

For [k=1 , k≤ni,k++,
  For[i=2,i≤n-1,i++,
    phi[[i]]=(phi[[i+1]]+phi[[i-1]]+x1[[i]]^2  h^2)/2;
    ];
  ];
(*calculate exact value*)
(*phiExact=x (1.0-x^3)/12 *)
phiExact=# (1.0-#^3)/12&
```

Below is to plot function

```
g1=ListPlot[Transpose[{x,phi}],Joined→True,PlotS
tyle-
>{{Blue,Thickness[.005]},{Green,Thickness[.005],D
ashing[{.02}]}},DisplayFunction->Identity];
g2=Plot[phiExact[x],{x,0,1},PlotStyle-
>{{Red,Thickness[.005]},{Green,Thickness[.005],D
ashing[{.02}]}},DisplayFunction->Identity];

ShowLegend[Show[g1,g2,PlotRange→All,DisplayFun
ction-
>$DisplayFunction],{{{Red,"Exact"},{Blue,"Estimate
"}},LegendPosition→{1.5,-0.5}}]
```

(* Finite Element * Poisson Type with right hand side x^2 *)

```
Clear[nd,c,ce,np,nd,ne,b,nl,x,val,ndp];
nd=301;
np=2;
ne=nd-1;
(* Input File *)
nl=Table[{i,i+1},{i,1,nd-1}];
delta=1/ne;
x=Table[(i-1) delta,{i,1,nd}];
val={0.0,0.0}
ndp={1,nd};
p={1,-1};
ce0=Outer[Times,p,p];
c=Table[0,{nd},{nd}];
b=Table[0,{nd}];
Qvec=Table[0,{nd}];
p=Table[0,{2}];
For[i=1,i≤ne,i++,
  knodes=nl[[i]];
  xl=x[[knodes]];
  l=xl[[2]]-xl[[1]];
  Qvec[[i]]=x[[i]]^2 l;
  (* Determine coefficient Matrix *)
  ce=ce0/l;
  (* Assemble *)
  For[j=1,j≤2,j++,
   ir=nl[[i,j]];
   iflag1=0;
   For[k=1,k≤np,k++,
    If[ir□ndp[[k]],
      c[[ir,ir]]=1;
      b[[ir]]=val[[k]];
      iflag1=1;
      ];
    ];
   If[iflag1□0,
    For[l=1,l≤2,l++,
      ic=nl[[i,l]];
      iflag2=0;
      For[k=1,k≤np,k++,
       If[ic□ndp[[k]],
         b[[ir]]-=ce[[j,l]]  val[[k]];
         iflag2=1;
         ];
       ];
      If[iflag2□0,
       c[[ir,ic]]+=ce[[j,l]]
       ];
      ];
    ];
   ];
  b[[i]]+=Qvec[[i]];
  ];
v=Inverse[c]. b;
(*Print["Node","    x    ","      v    "];
Print["---------------------------------------"];
Do[Print[i,"       ",x[[i]],"       ",v[[i]]],{i,1,nd}];*)
```

To visualize the function graph, the following expression is written

```
g1=ListPlot[Transpose[{x,v}],Joined-
>True,PlotStyle→{{Green,Thickness[.005]},{Green,T
hickness[.005],Dashing[{.02}]}},DisplayFunction→Id
entity]
```