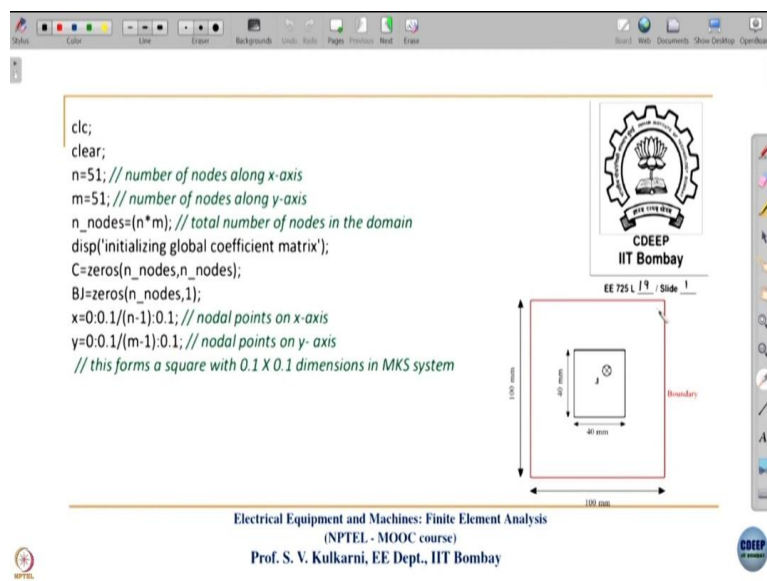


Electrical Equipment and Machines: Finite Element Analysis
Professor Shrikrishna V Kulkarni
Department of Electrical Engineering
Indian Institute of Technology Bombay
2D FEM Scilab Code: Manual Meshing
Lecture 19

Welcome to the 19th lecture. In the previous lecture, we saw FE procedure to solve 2D magnetostatic problems. Now we will see the coding aspects and manual meshing. We already saw a 1D code, so we are not going into line by line details of this code and only part of the code will be explained in detail.

(Refer Slide time: 00:42)



```
clc;
clear;
n=51; // number of nodes along x-axis
m=51; // number of nodes along y-axis
n_nodes=(n*m); // total number of nodes in the domain
disp('initializing global coefficient matrix');
C=zeros(n_nodes,n_nodes);
BJ=zeros(n_nodes,1);
x=0:0.1/(n-1):0.1; // nodal points on x-axis
y=0:0.1/(m-1):0.1; // nodal points on y-axis
// this forms a square with 0.1 X 0.1 dimensions in MKS system
```

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Here, the same geometry of a rectangular conductor enclosed in a boundary which is in the above slide is used in this code. This geometry is discretized into a 51×51 grid effectively. So the number of nodes along x and y axes are 51 and 51, and the total number of nodes in the grid will be $51 \times 51 = 2601$. Then we are initialising global coefficient matrices as given below.

```
C=zeros(n_nodes,n_nodes);
BJ=zeros(n_nodes,1);
```

All the entries of coefficient matrix (C) and source matrix (B) are made equal to 0. Then, coordinates of points on the x and y axes are calculated by using

```
x=0:0.1/(n-1):0.1; // nodal points on x-axis
y=0:0.1/(m-1):0.1; // nodal points on y-axis
```

(Refer Slide time: 02:00)

```

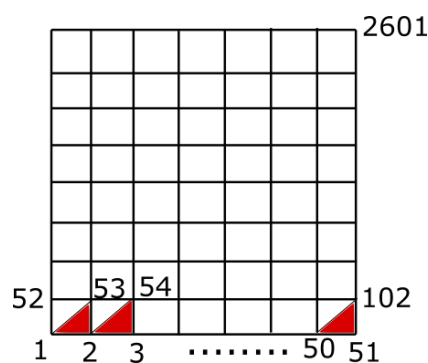
disp('forming p matrix');
// formation of p matrix
// Given domain is discretized into squares and then each square is
// divided into 2 triangular elements using diagonals of square

h=1; // h indicates node number (column index of p matrix)
// Indexing of node numbers and assigning coordinates for p matrix
for i=1:m // index for y
    for j=1:n // index for x
        // for each y, 51 x coordinates are formed
        p(1,h)=x(j);
        p(2,h)=y(i);
        h=h+1;
    end
end
end
p = [0 0.002 0.004 ... 0 0.002 ... 0.1
     0 0 0 ... 0.002 0.002 ... 0.1]

```

The slide also features a grid diagram with a shaded 'Conductor region' and node numbering. The x-axis nodes are labeled 1, 2, 3, ..., 50, 51. The y-axis nodes are labeled 1, 2, ..., 52, 53, ..., 102. A specific node is labeled (0,0,0)2551. The IIT Bombay logo and course information are also present.

After you have formed the grid on x-axis with nodes 1 to 51, then the nodes 52 to 102 and so on up to 2551 to 2601 are formed. The node numbers of the grid are shown in the figure in the above slide. Using the code in this slide we are forming the p matrix of the geometry. In an FE code, problem geometry is defined by using 3 matrices p, e and t. p matrix has information about coordinates of the nodes in the discretized domain. Now the problem domain for this geometry is discretized into squares (shown in the figure) and each square is divided into 2 triangular elements as shown in the following figure with one lower triangular element and one upper triangular element.



Then we are filling the p matrix whose entries are given below using the x and y coordinates of each node.

$$p = \begin{bmatrix} 1 & 2 & 3 & \dots & 52 & 53 & \dots & 2601 \\ 0 & 0.002 & 0.004 & \dots & 0 & 0.002 & \dots & 0.1 \\ 0 & 0 & 0 & \dots & 0.002 & 0.002 & \dots & 0.1 \end{bmatrix}$$

Using this code, we are populating the p matrix and you will get like the one shown in the above matrix. In this domain, the total number of nodes will be 2601 (= 51 × 51) so you will get the size of p matrix as 2 × 2601. The x and y coordinates of each of the nodes are saved in the p matrix. For example, for node number 1 (first column of p matrix) (x, y) is (0, 0), node number 2 (second column of p matrix) is (0.002, 0) and so on.

Remember in this lecture, we are doing manual meshing. In the next lecture, we will use gmsh which is a freeware meshing software and bookkeeping of element numbers and node numbers can be avoided by using this software. Here we are discussing manual meshing because your coding fundamentals will get consolidated in this lecture. Then we can go to gmsh when we start looking at complex problems. Also, we are not going into coding for each of those problems. By that time, we would have explained you at least 2 or 3 codes using which you can develop codes for other problems using such logics.

(Refer Slide time: 04:37)

The slide displays MATLAB code for forming a t matrix. The code includes comments explaining the formation of the matrix, the sub-domain number, and the global node numbers for each element. It uses a loop to iterate over elements and nodes, assigning values to the t matrix based on the element number and node indices.

The diagram shows a 51x51 grid with a triangular mesh. The nodes are numbered from 1 to 2601. The first three columns of the grid are highlighted in red, representing the first three elements. The t matrix is shown as a 4x4 matrix with the following structure:

1	1	1	...
1	2	3	...
2	3	4	...
53	54	55	...

The slide also includes the IIT Bombay logo and the text "CDEEP IIT Bombay".

Next, we will discuss about the t matrix. Now, this t matrix has information of subdomain numbers and global node numbers of each element. So, the size of this t matrix will be 4 × number of elements. The number 4 indicates the number of rows and each column represents an element, for example, column 1 denotes element 1 and the first entry (first row) of any column is the sub domain number and the 3 global node numbers of that element are in rows 2-4. The t matrix for the problem domain in this example is given below.

$$t = \begin{bmatrix} 1 & 1 & 1 & \dots \\ 1 & 2 & 3 & \dots \\ 2 & 3 & 4 & \dots \\ 53 & 54 & 55 & \dots \end{bmatrix}$$

The code in the above slide will form the lower triangular elements first. Here, we will first take all the lower triangular elements for the bottom-most layer (first layer) of elements then we go to the second row, again we form t matrix for lower triangular elements. Like this, we finish all the lower triangular elements and then we form the entries for t matrix of the upper triangular elements. So, using this code you can span the local triangular elements in the whole geometry. Here we are taking the same material everywhere. Actually, you have 2 materials in this problem geometry, one is the copper conductor and surrounding the conductor you have air. But both copper and air are non-magnetic with $\mu_r = 1$.

That is why we are defining all the elements with the same sub domain number. But actually, you could have named the conductor as sub-domain number 1 and air as sub-domain number 2. This will help to derive some physical quantity exclusively for the conductor using the code. For example, if you can calculate the energy associated with the conductor which is material number 1 or material number 2, then in that case you can define the 2 regions with different sub-domain numbers. But since we are using manual mesh and we already know the coordinates of each region and where it is lying, we do not have to differentiate the two regions separately by using 2 numbers. But in commercial software, for different materials although their properties maybe same, they are numbered by using different subdomain numbers. We could have done the same thing here also, but to reduce the coding effort we have named the subdomain numbers of all the elements by 1 only because the material property ($\mu_r = 1$) is same for both air as well as copper regions. Practically you have seen copper is diamagnetic material with μ_r slightly less than 1 but for engineering purposes, $\mu_r = 1$ for copper.

When you execute the subroutine explained in the above slide if $h = 1$ then the index is 1 and the following 3 commands will result into 1 2 and 53 and they are the global node numbers for the first element. Next time when you again run this loop with $h = 2$ then you will get 2 3 53 as the 3 node numbers. So, this way, you would have formed the t matrix entries corresponding to all the lower triangular elements in the whole domain.

(Refer Slide time: 08:30)

```

for i=1:m-1 //similarly, for upper triangular elements, 1st number: 2501
for j=1:n-1
t(1,h)=1;
t(2,h)=j+(i-1)*n;
// for i = 1, j = 1, h = 1, n = 51, t(2,1) = 1
t(3,h)=j+1+(i)*n; // t(3,1) = 53
t(4,h)=j+(i)*n; // t(4,1) = 52
h=h+1;
end
end

```

$t = \begin{bmatrix} 1 & \dots & 1 & 1 & \dots \\ 1 & \dots & 1 & 2 & \dots \\ 2 & \dots & 53 & 54 & \dots \\ 53 & \dots & 52 & 53 & \dots \end{bmatrix}$

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Similarly, we can form the t matrix entries for upper triangular elements which are given below.

$$t = \begin{bmatrix} 1 & \dots & 1 & 1 & \dots \\ 1 & \dots & 1 & 2 & \dots \\ 2 & \dots & 53 & 54 & \dots \\ 53 & \dots & 52 & 53 & \dots \end{bmatrix}$$

We already got the columns of t matrix entries before the column with entries 1 1 53 52, which are the entries for the lower triangular elements when you execute the similar code (in the above slide) with little bit different commands than the earlier one. When you run the subroutine for the first time you will get 1 1 53 and 52 as its output. Similarly, when you execute it next time you will get 1 2 54 53 and likewise you can form entries for the other elements also. So then at the end these 2 subroutines you would have got all the entries of the t matrix.

(Refer Slide time: 09:39)

`disp('.....');`
`[A3 n_elements]=size(t); // A3=4, n_elements=no. of columns of t (i.e., no. of elements)`
`//size of t matrix is 4 X no. of elements and 4 gets assigned to A3`
`[A1 n_nodes]=size(p); // A1=2, n_nodes=no. of columns of p (i.e., no. of nodes)`
`//size of p matrix is 2 X no. of nodes and 2 gets assigned to A1`
`disp('number of elements');`
`disp(n_elements); //for n = 51, m = 51, n_elements = 5000`
`disp('number of nodes');`
`disp(n_nodes); //for n = 51, m = 51, n_nodes = 2601`

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

`disp('.....');`
`disp('forming t matrix');`
`// formation of t matrix: sub-domain number`
`// and 3 global node numbers for each element`
`h=1; // here, h indicates element number (column index of t matrix)`
`// forming t matrix of lower triangular elements`
`for i=1:m-1`
`for j=1:n-1`
`t(1,h)=1; // same material everywhere`
`t(2,h)=j+(i-1)*n;`
`// for i = 1, j = 1, h = 1, n = 51, t(2,1) = 1`
`t(3,h)=j+1+(i-1)*n; // t(3,1) = 2`
`t(4,h)=j+1+(i)*n; // t(4,1) = 53`
`h=h+1; // incrementing the element number`
`end`
`end`

$$t = \begin{bmatrix} 1 & 1 & 1 & \dots \\ 1 & 2 & 3 & \dots \\ 2 & 3 & 4 & \dots \\ 53 & 54 & 55 & \dots \end{bmatrix}$$

Grid diagram showing a 51x51 grid with nodes numbered 1 to 51 along the bottom edge and 52 to 102 along the right edge. A yellow arrow points from the matrix to the grid.

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

We will go further. Now this command `[A3 n_elements]=size(t)` will give the size or dimensions of t matrix. The size of t matrix $4 \times$ number of elements as I explained will get assigned to A3 and number of elements which is equal to the number of columns of t matrix will get assigned to the variable n_elements by this command. Similarly, the command `[A1 n_nodes]=size(p)` will assign 2 to A1 because the size of the p matrix is $2 \times$ number of nodes. And the number of nodes will get assigned to n_nodes. And that is why and when you display those variables on the console, number of elements will be displayed as 5000 and number of nodes as 2601.

(Refer Slide time: 10:46)

```
//Forming element coefficient matrices
disp('forming element coefficient matrix');
for element=1:n_elements
    nodes=t(2:4,element);
    // 2nd, 3rd and 4th rows --> 3 global node numbers of each element
    //nodes' is a 3 X 1 matrix containing global
    // node numbers of the element under consideration
    Xc=p(1,nodes'); //x-coordinates of nodes, Xc is a 3 X 1 matrix
    Yc=p(2,nodes'); //y-coordinates of nodes, Yc is a 3 X 1 matrix
```

For element 1, nodes = [1 2 53]'

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Now we will see how element coefficient matrices are formed. When we run the for loop from 1 to number of elements and then we take in nodes. The three global nodes of each element get assigned to nodes matrix using the command `nodes=t(2:4,element)` because 2 3 4 rows of a column of the t matrix are the global node numbers of the element and the first row is the sub-domain number. By executing this command, we would get global node numbers of each element, for element number 1, entries of the nodes matrix are 1 2 and 53 which are the global node numbers.

Having got the global nodes of an element, we need to get x and y coordinates of those nodes. The x and y coordinates are stored in p matrix because it has information about the coordinates. By executing the 2 commands `Xc=p(1,nodes')` and `Yc=p(2,nodes')` the corresponding x and y coordinates of these 3 nodes can be obtained. Again, Xc and Yc will be 3×1 matrices corresponding to x and y coordinates of the 3 global node numbers of the corresponding element under consideration. So, for each element we will get this information which will be helpful to form element coefficient matrices which will see later.

(Refer Slide time: 12:35)

`P=zeros(3,1);`
`Q=zeros(3,1);`
`P(1)=Yc(2)-Yc(3);`
`P(2)=Yc(3)-Yc(1);`
`R(3)=Yc(1)-Yc(2);`
`Q(1)=Xc(3)-Xc(2);`
`Q(2)=Xc(1)-Xc(3);`
`Q(3)=Xc(2)-Xc(1);`

$$N_1 = \frac{1}{2\Delta} \left[(x_2 y_3 - x_3 y_2) + \underbrace{(y_2 - y_3)}_{P(1)} x + \underbrace{(x_2 - x_3)}_{Q(1)} y \right]$$

$$N_2 = \frac{1}{2\Delta} \left[(x_3 y_1 - x_1 y_3) + \underbrace{(y_3 - y_1)}_{P(2)} x + \underbrace{(x_3 - x_1)}_{Q(2)} y \right]$$

$$N_3 = \frac{1}{2\Delta} \left[(x_1 y_2 - x_2 y_1) + \underbrace{(y_1 - y_2)}_{P(3)} x + \underbrace{(x_1 - x_2)}_{Q(3)} y \right]$$

Ref: M. N. O. Sadiku and S. V. Kulkarni, *Principles of Electromagnetics*, Sixth Edition, Oxford University Press, India, 2015
 (Asian adaptation of M. N. O. Sadiku, *Elements of Electromagnetics*, Sixth International Edition, Oxford University Press)

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

We have seen the expressions that are coded in the above slide earlier and shape functions N_1 , N_2 and N_3 are given by the three equations in the above slide. Now we call $y_2 - y_3$ as P_1 , $x_2 - x_3$ as Q_1 and similarly the other 2 sets of expressions as indicated in the slide. Then we will code the expressions of P_1 to Q_3 as given in the above slide to simplify our equations of the entries of local coefficient matrices. As I said earlier, these expressions would be required quite often when we develop an FE code.

(Refer Slide time: 13:23)

```

//finding x and y coordinates of the centroid
//see if the element lies within the rectangular conductor
cX = sum(Xc)/3;
cY = sum(Yc)/3;
if (cX<=0.07 && cX>=0.03) && (cY<=0.07 && cY>=0.03) then
  Mu(element) = 4*%pi*1e-7;
  bj = 1e3; //J-> current (amp-turn) density
else
  Mu(element) = 4*%pi*1e-7;
  bj = 0;
end
  
```

(0,0) (0.07,0.07) (0.1,0.1)
 (0.07,0.07)
 (0.07,0.03)

Conductor dimensions 0.04 x 0.04

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Now in our geometry we have a rectangular conductor and an air box as shown in the figure of the above slide. As far as the material properties are concerned both air and copper are non-magnetic $\mu_r = 1$ but what is the main difference between the 2 regions. There is current in the

conductor region and we have to impose it for this region. So, we have to find all the elements which lie within this conductor so that we can assign current density and the corresponding b matrix for those elements can be derived.

The x and y coordinates of the centroid of an element is calculated by using the following commands.

$$\begin{aligned}cX &= \text{sum}(Xc)/3; \\cY &= \text{sum}(Yc)/3;\end{aligned}$$

Then we know that for this conductor, the coordinate of the leftmost point and the rightmost point, that is top most point on the right-hand side and bottom most point on the left side are (0.07, 0.07) and (0.03, 0.03) respectively, which are indicated in the figure on the above slide. Remember we had the whole geometry from 0 to 100 mm whereas the conductor was from 30 to 70 mm in x and y directions. So in this code we are checking whether the x and y coordinates of the centroid of the element lie between 0.03 and 0.07 and if it is true, then that element is within the conductor rectangular area.

For example, let us say we have an element like the one which is indicated in the above figure. Now, we are calculating the centroid of that element which is at the centre of the triangle. If that centroid lies within the conductor region, then the triangle is lying within the conductor. And if that being the case then we assign $\mu = \mu_r \mu_0 = 4\pi \times 10^{-7}$ because $\mu_r = 1$ and then $b_j = 10^3$. This we are taking as an example and we are considering current density in the conductor as 10^3 A/m^2 .

It should be noted that it can be current density or ampere turn density in case of a winding. Now the present case is like a single conductor which is like a 1 turn winding. So you have to feed current as ampere turn density. When you solve a transformer or a rotating machine example, there we will define ampere turn density in the current carrying areas. And it is equal to ampere turns divided by the corresponding cross-sectional area of the winding. So current density is fed in the conductor region and if the element is outside the conductor area then again, $\mu_r = 1$ but $b_j = 0$ because there is no current carrying part in the air region.

(Refer Slide time: 16:52)

// area of triangular elements
 $\text{delta} = 0.5 * \text{abs}((P(2) * Q(3)) - (P(3) * Q(2)))$;
 //Absolute value is taken since
 //the three nodes may not have been numbered in the anticlockwise direction

$$\Delta = \frac{1}{2} [(y_1 - y_2)(x_2 - x_1) - (y_1 - y_3)(x_3 - x_1)]$$

$$= \frac{1}{2} [x_2 y_1 - x_2 y_2 - x_1 y_2 + x_1 y_3 + x_1 y_2 + x_1 y_1 - x_3 y_2 - x_3 y_1 + x_3 y_3]$$

$$\Delta = \frac{1}{2} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} = \frac{1}{2} [1(x_2 y_3 - y_2 x_3) - 1(x_3 y_1 - y_3 x_1) + 1(x_1 y_2 - y_1 x_2)]$$

$$\Delta = \frac{1}{2} [x_2 y_3 - y_2 x_3 - x_3 y_1 + y_3 x_1 + x_1 y_2 - y_1 x_2]$$

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

We have already seen the expression of the command $\text{delta} = 0.5 * \text{abs}((P(2) * Q(3)) - (P(3) * Q(2)))$ in the above slide which calculates the area of the triangle. This expression can be verified by using the derivations in the above slide. We have already got the expressions for P_i s and Q_i s. Also, absolute value is taken, because, in case, if the nodes are not numbered in the anti-clockwise direction then area will be negative. We have explained the expressions in the above slide earlier in one of the previous slides. We verified this expression for the area of a triangle in terms of the determinant and coefficients P_i s and Q_i s.

(Refer Slide time: 17: 37)

```

for i=1:3
  for j=1:3 // 3 X 3 element coefficient matrix
    c(element,i,j)=(P(i)*P(j))+(Q(i)*Q(j))/(4*delta*Mu(element));
  end
end // 3 X 1 element level source matrix
be(element,:)=bj*(delta/3)*[1;1;1];
end
  
```

$$c_{ij}^e = \frac{1}{\mu^e} \int_{S^e} \nabla N_i \cdot \nabla N_j dS^e = \frac{1}{4\Delta\mu^e} [P_i P_j + Q_i Q_j]$$

$$B^e = \begin{bmatrix} b_1^e \\ b_2^e \\ b_3^e \end{bmatrix} = J \Delta \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Electrical Equipment and Machines: Finite Element Analysis
 (NPTEL - MOOC course)
 Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Now, we form element coefficient matrices. The entries of the element coefficient matrix can be formed by the following command.

$$c(\text{element},i,j)=((P(i)*P(j))+(Q(i)*Q(j)))/(4*\text{delta}*Mu(\text{element}));$$

It is so simple in terms of coding effort. Also, it is very straightforward. We have seen the following expression of c_{ij}^e .

$$c_{ij}^e = \frac{1}{\mu^e} \int_{S^e} \nabla N_i \cdot \nabla N_j dS^e = \frac{1}{4\Delta\mu^e} [P_i P_j + Q_i Q_j]$$

Remember that this command is being run in the for loop for each element. Suppose for element number 1, we will get the corresponding 3x3 matrix. These matrices for all the elements are stored in a single variable as stacks. Mr. Sairam has explained about this thing earlier.

3x3 matrices of elements 1, 2, 3, and so on are stacks and that would get formed when you run the for loop for each element. For example, for the first element, i and j goes from 1 to 3 and the corresponding 3x3 element coefficient matrix is formed. The corresponding source contribution to that element will be given by the command $be(\text{element},:) = bj*(\text{delta}/3)*[1;1;1]$ which is given by the following expression. ;

$$B^e = \begin{bmatrix} b_1^e \\ b_2^e \\ b_3^e \end{bmatrix} = J \Delta \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

If the current density exists in that element, it basically gets apportioned to each of the nodes of that triangular element equally as $\frac{J\Delta}{3}$. That is what is being coded in the above slide.

(Refer Slide time: 19:44)

```

disp('forming global coefficient matrix');
for element=1:n_elements
    nodes=t(2:4,element);
    // Formation of the global coefficient matrix
    for i=1:3
        for j=1:3
            C(nodes(i),nodes(j))=C(nodes(i),nodes(j))+c(element,i,j);
        end
        B(nodes(i),1) = be(element,i) + B(nodes(i),1);
    end
end
nodes = [1 2 53]^T
for i = 1, j = 1
    C(1,1) = C(1,1) + c(1,1,1)
    C(1,2) = C(1,2) + c(1,1,2)
    C(1,3) = C(1,3) + c(1,1,3)
for i = 1, j = 2
    C(1,2) = C(1,2) + c(1,1,2)
for i = 1, j = 3
    C(1,3) = C(1,3) + c(1,1,3)
for i = 1, j = 1
    C(1,1) = C(1,1) + c(2501,1,1)
    C(1,2) = C(1,2) + c(2501,1,2)
    C(1,3) = C(1,3) + c(2501,1,3)
for i = 1, j = 2
    C(1,2) = C(1,2) + c(2501,1,2)
for i = 1, j = 3
    C(1,3) = C(1,3) + c(2501,1,3)
nodes = [1 2 53]^T
for i = 1
    B(1) = B(1) + be(1,1)
    B(2) = B(2) + be(1,2)
    B(3) = B(3) + be(1,3)
nodes = [1 53 52]^T
for i = 1
    B(1) = B(1) + be(2501,1)
    B(53) = B(53) + be(2501,2)
    B(52) = B(52) + be(2501,3)

```

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

```

disp('-----');
disp('forming t matrix');
// formation of t matrix: sub-domain number
// and 3 global node numbers for each element
h=1; // here, h indicates element number (column index of t matrix)
// forming t matrix of lower triangular elements
for i=1:m-1
    for j=1:n-1
        t(1,h)=1; // same material everywhere
        t(2,h)=j+((i-1)*n);
        // for i = 1, j = 1, h = 1, n=51, t(2,1) = 1
        t(3,h)=j+1+((i-1)*n); // t(3,1) = 2
        t(4,h)=j+1+((i)*n); // t(4,1) = 53
        h=h+1; // incrementing the element number
    end
end
end

```

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Going further, like we saw in case of the 1D FE code, we will see how to get the global coefficient matrix by combining all element coefficient matrices. Again, we will run a for loop from 1 to the number of elements. For each element, we take element coefficient matrices and go on appending them to the global coefficient matrix. Each element will have 3 nodes and some of the nodes and edges of each element may be common to different elements. By execution of the command `nodes=t(2:4,element)` for each element, we get its global node numbers using the t matrix into the 3×1 column matrix named as nodes.


Then we run the following for loops in which i goes from 1 to 3, and j goes from 1 to 3.

```

for i=1:3
    for j=1:3
        C(nodes(i),nodes(j))=C(nodes(i),nodes(j))+c(element,i,j);
    end
    BJ(nodes(i),1) = be(element,i)+ BJ(nodes(i),1);
end

```

These for loops will read the 9 entries from the element coefficient matrix. Depending upon the corresponding global node number, each of those 9 entries will get appended to the corresponding entries of the global coefficient matrix because `nodes(i)`, `nodes(j)` will return the global number from the nodes matrix. For example, element 1 is a lower triangular element with global nodes 1, 2, and 53. When we run these two for loops with $i = 1$ and $j = 1$, $c(1,1,1)$ gets appended to $C(1,1)$ as given in the following figure. Because the local node number 1 of element 1 is global node number 1.

Element 1 \rightarrow 

nodes = $[1 \ 2 \ 53]^T$

for $i = 1, j = 1$ element

$$C(1,1) = \underbrace{C(1,1)}_{=0} + c(1,1,1)$$

\uparrow
 $i \ j$

So that is how the contribution of node 1 in element 1 will get added to global coefficient matrix. You can see here that the initial value of $C(1,1)$ is 0 because for the first time we are appending something to the $C(1,1)$ position of global coefficient matrix. So $C(1,1)$ will get filled with $c(1,1,1)$ ((1,1) entry of the element coefficient matrix of element 1).


Similarly, for element 1, when $i = 1$ and $j = 2$, $c(1,1,2)$ of element 1 will get appended to $C(1,2)$ as shown in the figure below. Because again the global node number of this local node 2 is 2.

for $i = 1, j = 2$

$$C(1,2) = \underbrace{C(1,2)}_{=0} + c(1,1,2)$$

When $i = 1$ and $j = 3$ for the first element, the global node number of local node 3 for the element 1 is 53. So $c(1,1,3)$ of element 1 will be appended to $C(1,53)$. The (1,3) position of local coefficient matrix of element 1 will correspond to (1,53) position of global coefficient matrix. So $C(1,53)$ will get the value of $c(1,1,3)$ of element 1.

Now you go to element number 2501 which is the first upper triangular element that comes after going through all lower triangular elements. And the global node numbers for this element are 1, 53, and 52. Now for $i = 1$ and $j = 1$, $C(1,1) \neq 0$ because this node number was already encountered for element 1. We will append $c(2501,1,1)$ to $C(1,1)$ as shown below.

Element 2501 \rightarrow 

nodes = $[1 \ 53 \ 52]^T$

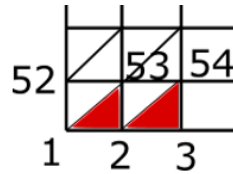
for $i = 1, j = 1$ element

$$C(1,1) = \underbrace{C(1,1)}_{\neq 0} + c(2501,1,1)$$

\uparrow
 $i \ j$

Now, the global node number 1 is common to element 1 (lower triangular element) and element 2501 (upper triangular element). So that is why $c(1,1,1)$ and $c(2501,1,1)$ will be added in the formation of the global coefficient matrix. That is why $C(1,1)$ will be contributed by (1,1) of element number 1 and (1,1) of element 2501. The global node number 53 will be common to

6 elements because this node is in the inner region of the grid. You can see in the following figure, the node 53 will be common to 6 triangular elements.



Since the global node number 53 is common to 6 elements, the diagonal entry $C(53,53)$ will get appended 6 times. But this whole thing will be done using the following statement.

$$c(\text{element},i,j)=((P(i)*P(j))+(Q(i)*Q(j)))/(4*\text{delta}*\text{Mu}(\text{element}))$$

You may feel that it is so complicated, but it is just happening due to the execution of the above statement. Similarly, now we consider the b matrix (source matrix), which is a 1-dimensional matrix. For element number 1, $b_e(1,1)$ of local node number 1 of element 1 will get appended to the global BJ(1). Initially, it is 0 and for the first time it will get updated by the contribution of element 1.

For element 2501, global node number 1 will be encountered for the second time. The corresponding contribution of that element will get appended to BJ(1). And this goes on and by the end of this we would have got both global matrices C, BJ. We are calling the source matrix as BJ and not B. Because this has the contribution of only current source. Another contribution from boundary conditions will be there and that matrix will be modified when we impose them. That is why we are calling this matrix as BJ.

(Refer Slide time: 27:35)

```

disp('applying boundary conditions');
// e matrix is not formed hence boundary conditions are applied manually
// 1st edge, bottom edge
for i=1:n
    C(i,:)=zeros(1,n_nodes);
    C(i,i)=1;
    BJ(i,1)=0;
end

// 2nd edge, vertical left side edge
for i=n+1:n:(n*m)-2*n+1
    C(i,:)=zeros(1,n_nodes);
    C(i,i)=1;
    BJ(i,1)=0;
end
    
```

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Now, we apply the boundary conditions. For the outermost boundary, we are specifying magnetic vector potential $A = 0$. For all the nodes which are on the outer boundary of the domain we have to specify $A = 0$. Although it is a vector potential, here, since it is a 2D analysis, it is a scalar because its direction is already fixed. A is only in the z direction. Since, the direction of A is already known, we need to determine only the magnitude of A at every node in the domain. Because we have converted the problem domain wherein we need to calculate potential at every point to a problem where potentials are calculated only at the grid points.

Once you calculate the potentials at grid points, then potential at any point within an element can be determined by using the formula $A = a + bx + cy$ which is our original approximation. The way we imposed the boundary conditions in the above slide is also explained to you when we saw the 1D code. The boundary conditions are imposed by making the corresponding diagonal entry of the node as 1 and off-diagonal entries as 0.

The easiest way to do this is first you make all the entries of the row that corresponds to a boundary node as 0 and then make the diagonal entry 1, rather than searching which is the diagonal entry. This way of imposing boundary conditions is straightforward. For the row of an i^{th} boundary node, all entries are made 0 and the diagonal entry $C(i,i)$ is made 1 by the following commands.

```
C(j,:)=zeros(1,n_nodes);  
C(i,i)=1;
```

In the last lecture, we saw an example of imposing $A_5 = 0$ in 1D code. In that example, we made the diagonal entry of 5th row as 1 and on the right-hand side, we made the 5th entry of B as 0. All the diagonal entries are already 0 there.

The same thing we repeat for the bottom edge by using the first for loop in the above slide which runs from 1 to 51 because n and m are equal to 51. Then for the second vertical edge we are running the second for loop in the above slide from 52 to 2500 with a step size of 50 because 1 is already assigned in boundary edge. Since this is manual meshing, you have to do this bookkeeping of boundary nodes. In the next lecture, we see the use of freeware called gmsh, there all these things become very easy. We do not have to worry about keeping the track of boundary node numbers to impose boundary conditions.

(Refer Slide time: 30:59)

```

// 3rd edge, vertical right side edge
for i=2*n:n:(n*m)
    C(i,:)=zeros(1,n_nodes);
    C(i,i)=1;
    BJ(i,1)=0;
end

// 4th edge, top edge
for i=(n*(m-1)+1):(n*m)-1
    C(i,:)=zeros(1,n_nodes);
    C(i,i)=1;
    BJ(i,1)=0;
end
    
```

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Then the same procedure that we use for this bottom edge is followed for the second vertical edge and finally for the top edge. Only difference is that the index for i is adjusted so that we get proper numbers of nodes on the edges when we execute the for loops.

(Refer Slide time: 31:19)

```

disp('computing nodal potentials');
A=C\BJ;

//Here, A is a column vector - it is converted into n X m matrix,
//similarly, two rows of p matrix (x and y coordinates) are converted
//into n X m matrix
Vn=matrix(A(1:n*m),n,m);
//potentials at each node of n X m grid
XX=matrix(p(1,1:n*m),n,m);// x coordinates of n X m grid
YY=matrix(p(2,1:n*m),n,m);// y coordinates of n X m grid
    
```

$CA = B$

reshape is the command used in Matlab® to convert a column vector into an $n \times m$ matrix

Electrical Equipment and Machines: Finite Element Analysis
(NPTEL - MOOC course)
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

Now we are nearing the end of this code. Although we are calling the right hand side matrix as BJ, it got modified with the boundary conditions. Now, this matrix has got the contributions from both source (J) and boundary conditions. Now, we have the matrix equation of the form $CA = B$. We have seen this earlier and then we have to invert C. So A is $\text{inv}(C)B$. Now, in both Matlab and Scilab, $\text{inv}(C)B$ is calculated by using $C \setminus B$.

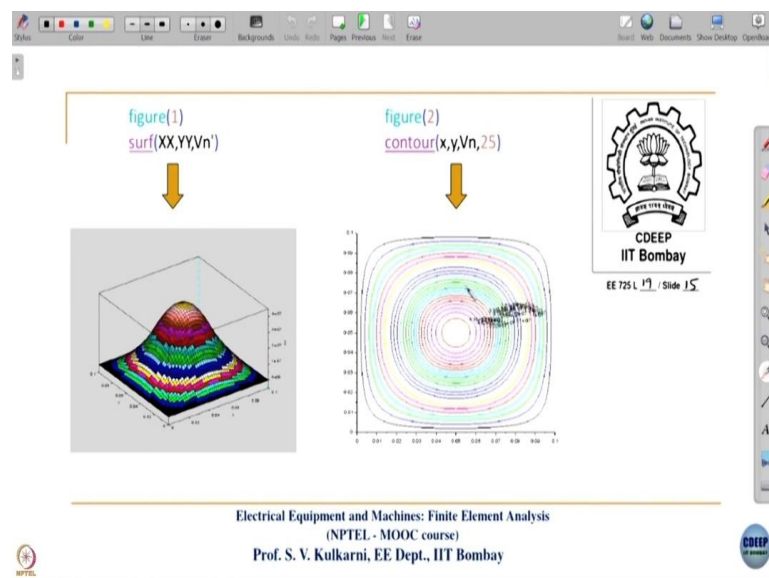
Now, we have got the solution (nodal potential values) in the matrix A. By executing the command $(A=C\backslash B)$ we have got unknown magnetic vector potential values at all points within the geometry. At the boundary points we will get the potentials that we have already imposed. Now we have to convert this A which is a column vector into a 2 dimensional matrix because the problem geometry is 2 dimensional. It is of no use for us to have a column vector for the plotting purpose because we need to plot it in the 2 dimensional space. So that is why we are converting this column vector into an $n \times m$ matrix. And similarly the two rows of p matrix which are x, y coordinates of nodes in the problem domain has to be converted into an $n \times m$ matrix. That means at each of the nodes, we have x and y coordinates in p matrix.

So, we have to convert them as 2 dimensional matrices for each of those x and y coordinates in the 2 dimensional plane. Now, we have to plot the corresponding A values, which are also in 2 dimensional matrix form. These are accomplished by using the following commands.

```
Vn=matrix(A(1:n*m),n,m); //potentials at each node of n X m grid
XX=matrix(p(1,1:n*m),n,m); // x coordinates of n X m grid
YY=matrix(p(2,1:n*m),n,m); // y coordinates of n X m grid
```

A similar command in Matlab is 'reshape'. Those who are having Matlab commercial software, they can use the same code with using 'reshape' instead of 'matrix'.

(Refer Slide time: 34:01)



So, having done that now we need to plot the potential values using the following commands for surface and contour plots which are shown in the above slide.

`surf(XX,YY,Vn')`
`contour(x,y,Vn,25)`

By observing the above plots, you can remember that $A = 0$ at the boundaries and A will be higher at the centre of the conductor. So, this is how you can plot the potentials. We will end this 19th lecture here and then we will see how the same coding can be done using the mesh generated using gmsh software, which is a freeware. Thank you.

(Refer Slide time: 34:49)

L19: Review Question

Q: In the demonstrated 2D code, subdomain number 1 is specified to both conductor and air regions. Modify the code to specify subdomain number 1 for the air region and subdomain number 2 for the conductor region.

