Electrical Equipment and Machines: Finite Element Analysis

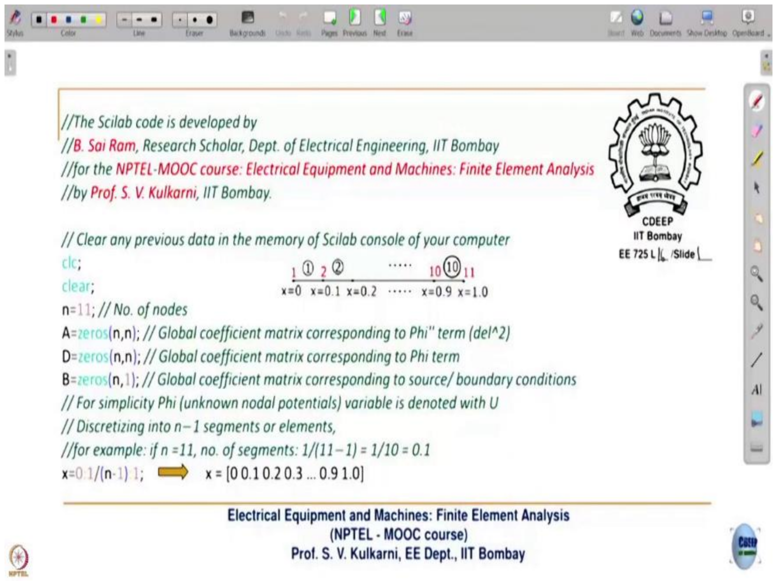
Professor Shrikrishna V. Kulkarni
Department of Electrical Engineering
Indian Institute of Technology Bombay

Lecture 16

1D FEM: Scilab Code

Welcome to the 16th lecture. As we discussed in the previous lecture, we will be developing an FEM code in Scilab. Scilab is a freeware that can be downloaded and installed. Also, its syntax is similar to MATLAB. So, the same code can be used in MATLAB as well. The codes that we are demonstrating in this lecture and the course can be used in MATLAB also.

(Refer Slide Time: 00:53)



```
//The Scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the NPTEL-MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of Scilab console of your computer
clc;
clear;
n=11; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to source/ boundary conditions
// For simplicity Phi (unknown nodal potentials) variable is denoted with U
// Discretizing into n-1 segments or elements,
//for example: if n =11, no. of segments: 1/(11-1) = 1/10 = 0.1
x=0:1/(n-1):1;
```

$\Rightarrow$ x = [0 0.1 0.2 0.3 ... 0.9 1.0]

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

Going further into the code, as mentioned in the previous lecture, the steps involved in an FE code will start with discretization of the problem geometry. So, first, we will discuss how to discretize a 1-dimensional geometry. In the above slide you can see, the first two lines in the code are clc and clear.

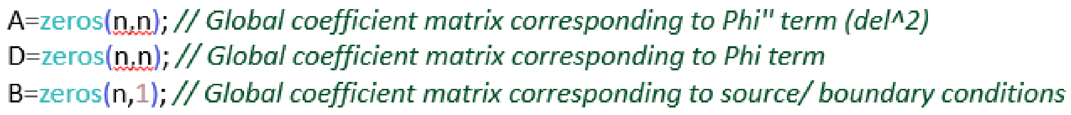
What is the use of these two lines? You will be using this scientific computing software very often. Data of some previous code may be saved in the console and that has to be erased. Otherwise, if the variable of the previous code and the present are common, they may interfere with each other and that may give you some errors in the results. That is why at the start of every code, you have to add these 2 statements (clc and clear) which will make your results safe.

The next step is discretization. A 1-dimensional geometry is a straight line and can be divided into segments. They are defined by its starting and ending nodes. So, the discretization of a one-dimensional domain involves nodes and segments.

One more point is that the algorithm that we are discussing in this lecture is not the only way to develop the code for this problem. In a number of ways one can develop the code for a problem. The procedure that we are going to discuss in this lecture is one way to code. Here, we start with discretization by defining n = 11, where n is the number of nodes. While discretizing the 1-dimensional domain, you can start with either defining number of segments or number of nodes. These two algorithms will be different. Here, we are considering the algorithm which starts with choosing the number of nodes.

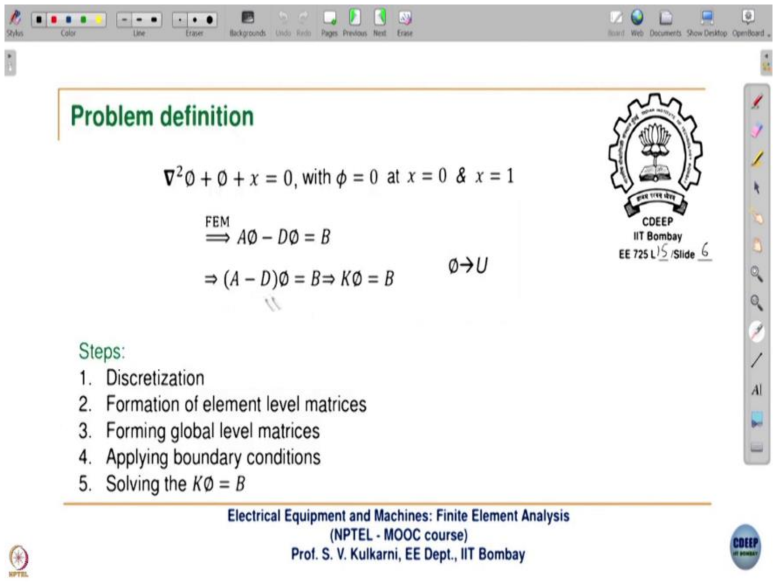
The 11 nodes will discretize the 1-dimensional geometry into 10 segments. In 1-dimensional geometry which is a straight line, if you have n number of nodes, they will divide the geometry into $n - 1$ segments. In the previous lecture also, you can recall that there were 4 nodes and number of elements were 3 and this is a simple logic.

Once you define the number of nodes, that will automatically defines the number of segments. So the number of segments will be $n - 1$. So, for a 1D geometry , this is how the discretization will be done. And then, we are defining the global coefficient matrices as given below



```
A=zeros(n,n); // Global coefficient matrix corresponding to Phi" term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to source/ boundary conditions
```

(Refer Slide Time: 03:55)



Problem definition

$\nabla^2\emptyset + \emptyset + x = 0$, with $\phi = 0$ at $x = 0$ & $x = 1$

$\overset{\text{FEM}}{\Longrightarrow} A\emptyset - D\emptyset = B$

$\Rightarrow (A - D)\emptyset = B \Rightarrow K\emptyset = B$ $\quad \emptyset \rightarrow U$

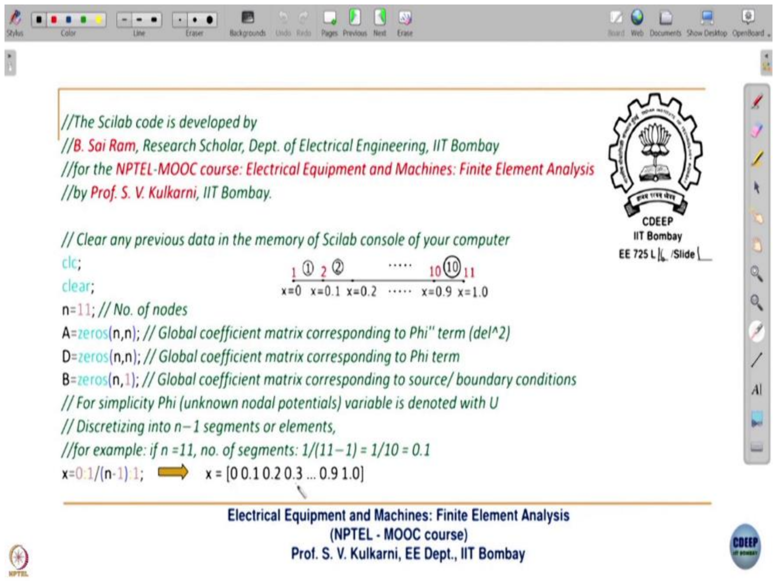
Steps:
1. Discretization
2. Formation of element level matrices
3. Forming global level matrices
4. Applying boundary conditions
5. Solving the $K\emptyset = B$

CDEEP
IIT Bombay
EE 725 L 15 /Slide 6

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

In the matrix equation, which is shown in the above slide of the previous lecture, there are 3 matrices, A, D, B. So now our job is to fill these matrices. These 3 matrices are defined as given in the above lines.

(Refer Slide Time: 04:05)



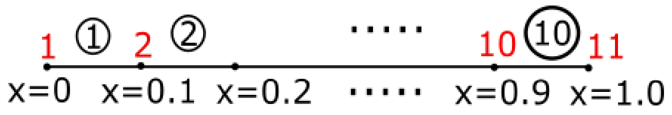
```
//The Scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the NPTEL-MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of Scilab console of your computer
clc;
clear;
n=11; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to source/ boundary conditions
// For simplicity Phi (unknown nodal potentials) variable is denoted with U
// Discretizing into n-1 segments or elements,
//for example: if n =11, no. of segments: 1/(11-1) = 1/10 = 0.1
x=0:1/(n-1):1;
```

$\Rightarrow$ x = [0 0.1 0.2 0.3 ... 0.9 1.0]

CDEEP
IIT Bombay
EE 725 L 16 /Slide 1

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

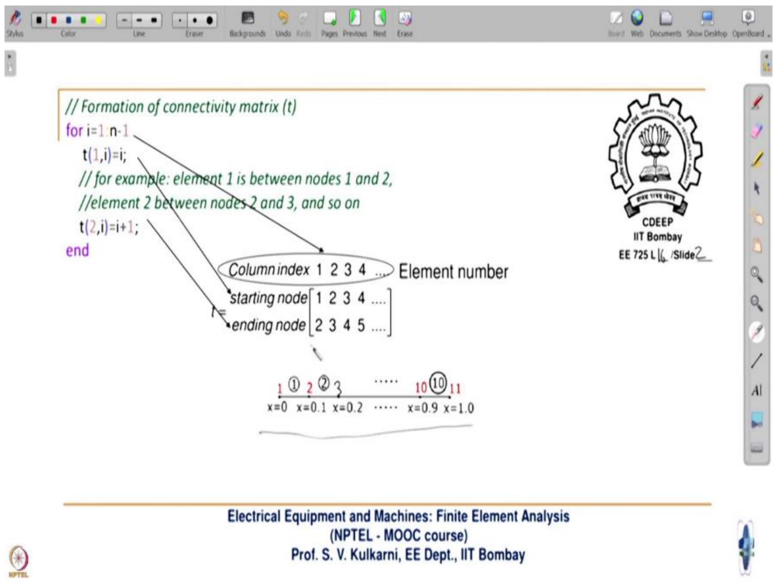
We are defining A = zeros(n,n), (n,n) defines the dimension of the matrix and zeros will make the matrix as 0 completely. Now that matrix will be updated with the entries of element coefficient matrices. As we have seen in the previous lecture, the size of of A and D matrices will be $n \times n$. That is why we have used (n,n) while defining A matrix. In case of the source matrix (B), it is defined with (n,1), also you can correlate these matrices directly with previous lecture's matrices. Next step in the code is finding the coordinates of the nodes in the discretized domain which is as shown in the following figure.



The nodes of the discretized domain and their positions are indicated in the above figure. Now the question is how to calculate these positions. In this example, we are choosing the coordinates or the points to be equispaced as shown in the above figure because this will simplify the discretization procedure. You can also choose nodes that are non equispaced with more number of nodes at the starting region and end region and less number of nodes in the middle region of the domain. But that will complicate the part of your code that involves choosing coordinates.

That is why, for simplicity, equispaced nodes are chosen. Let us say in this case for n = 11, the command x = 0:1/n:1 will divide the domain and give the points in between 0 and 1, with a step size of 1/(n-1). This command will result in to a row matrix x whose entries are starting from 0 to 1 with a step size of 0.1 for the present discretization with 11 nodes.

(Refer Slide Time: 06:12)



```
// Formation of connectivity matrix (t)
for i=1:n-1
    t(1,i)=i;
   // for example: element 1 is between nodes 1 and 2,
   //element 2 between nodes 2 and 3, and so on
   t(2,i)=i+1;
end
```

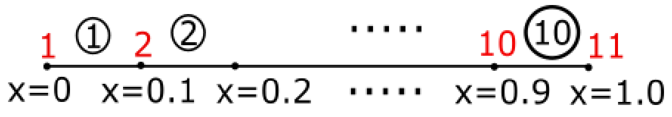
$$t = \begin{array}{r|cccc l} \text{Column index} & 1 & 2 & 3 & 4 & \dots \\ \hline \text{starting node} & 1 & 2 & 3 & 4 & \dots \\ \text{ending node} & 2 & 3 & 4 & 5 & \dots \end{array}$$

Element number

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

Now once we determined the details of nodes and their positions, we need to know how these nodes are connected. In the case of a 1-dimensional domain, it is obvious that they are connected by a straight line. But in the case of 2D, it will be difficult so that is why we frame a t matrix. In the present 1D problem also we are using it to make you familiar about t matrix.

t matrix is called as connectivity matrix and it contains information about how each node is connected to other nodes. So you can see the discretized domain in the following figure. The numbers which are encircled in the figure are element numbers.

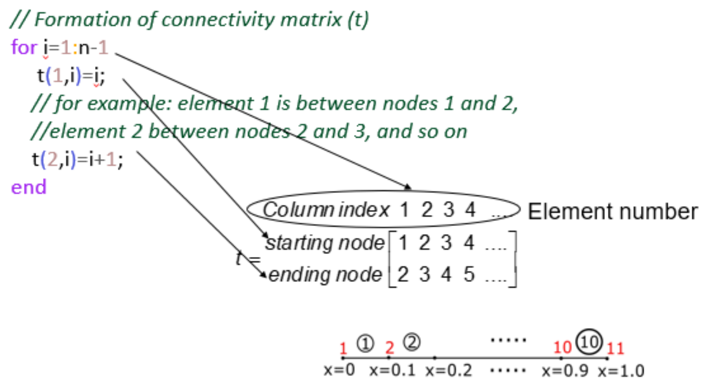


For element number 1, the starting node is 1 and the ending node is 2, for the second element, the starting node is 2 and the ending node is 3 and for the third element, the starting node is 3, and the ending node is 4. What is the common thing here? The element number is same as its starting node number. This information will simplify steps involved in the code.

So, assigning the element number of each element to its starting node number and the ending number is its element number plus 1. That is what we have coded using the following for loop in the above slide. As shown in the above slide in the *t* matrix for each element, the starting node number is its element number which is denoted by i.

The ending node number is 1 plus the corresponding element number. So, with the for loop in the above slide, for every *i* (element number), we will get the starting node number of the element in the first row. The definitions of matrices in Scilab and MATLAB are same in both software. The first index of a matrix is row number and the second index is column number. In the code, *i* in the *t* matrix is going to the column number. That means element number is nothing but the column number of the t matrix. And the first row will denote the starting node number and the second row will indicate the ending node number which is 1 plus element number.

Now, why this matrix is called a connectivity matrix? Where is the connectivity information stored in this matrix? You can understand this from the matrix that is formed using the following code.



```
// Formation of connectivity matrix (t)
for i=1:n-1
    t(1,i)=i;
   // for example: element 1 is between nodes 1 and 2,
   //element 2 between nodes 2 and 3, and so on
   t(2,i)=i+1;
end
```

Column index 1 2 3 4 ...
Element number
t =
starting node [1 2 3 4 ....]
ending node [2 3 4 5 ....]
1 ① 2 ② ..... 10 ⑩ 11
x=0 x=0.1 x=0.2 ..... x=0.9 x=1.0

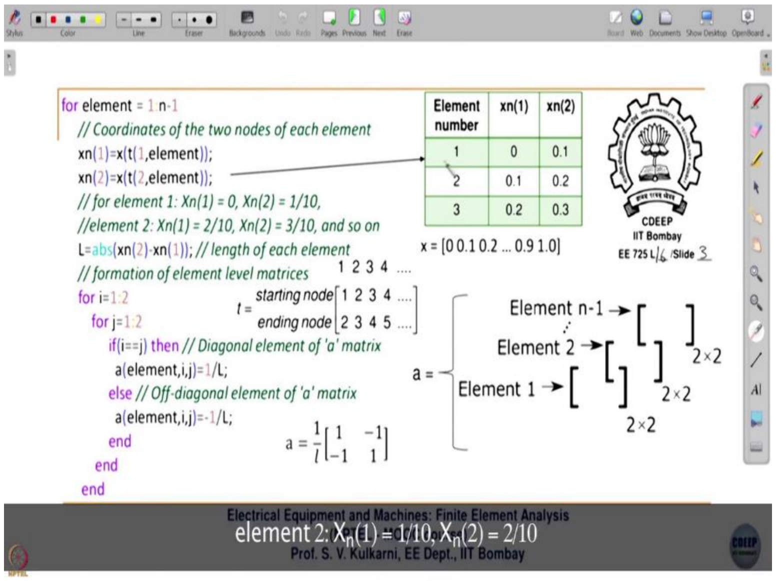
For a 1D geometry, if you run the for loop in the above figure, we will get the *t* matrix like the one shown in the above figure, where its column numbers are element numbers, the first row is the starting node number and the second row is the ending node number of the element.

The entries in the *t* matrix are 1, 2 in the first column, 2, 3 in the second column, 3, 4 in the fourth column, etc. Now, by looking at the matrix which is in the above figure, we can know the connectivity. How? The ending node of the first element is the starting node of the second element. So, through the second node, the first and second elements are connected. Similarly, through the third node, the second and third elements are connected, and so on. Such information is stored in the *t* matrix .

The *t* matrix plays a very important role in converting element coefficient matrices to the global coefficient matrix. We will be using this matrix very often. So, we need to know each and every part of the connectivity matrix (t). And another point to be highlighted is that the first row indicates the local node number 1 and the second row indicates the local node number 2.

That means, for element 1, local node number 1 is global node 1, local node number 2 is global node 2. Similarly, for element 2, the local node number 1 is global node 2, and local node number 2 is global node 3. The first row of t matrix indicates the first local node number of each element and the second row of *t* matrix indicates the second local node of all elements.

(Refer Slide Time: 10:46)



```
for element = 1:n-1
  // Coordinates of the two nodes of each element
  xn(1)=x(t(1,element));
  xn(2)=x(t(2,element));
  // for element 1: Xn(1) = 0, Xn(2) = 1/10,
  //element 2: Xn(1) = 2/10, Xn(2) = 3/10, and so on
  L=abs(xn(2)-xn(1)); // length of each element
  // formation of element level matrices
  for i=1:2
    for j=1:2
      if(i==j) then // Diagonal element of 'a' matrix
        a(element,i,j)=1/L;
      else // Off-diagonal element of 'a' matrix
        a(element,i,j)=-1/L;
      end
    end
  end
```

Element number	xn(1)	xn(2)
1	0	0.1
2	0.1	0.2
3	0.2	0.3

$x = [0\ 0.1\ 0.2 \ldots 0.9\ 1.0]$

$$t = \begin{array}{r} \\ \text{starting node} \\ \text{ending node} \end{array} \begin{array}{c} 1\ 2\ 3\ 4\ \ldots \\ \begin{bmatrix} 1 & 2 & 3 & 4 & \ldots \\ 2 & 3 & 4 & 5 & \ldots \end{bmatrix} \end{array}$$

$$a = \frac{1}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$a = \left\{ \begin{array}{l} \text{Element } n-1 \rightarrow [\ \ ]_{2\times 2} \\ \vdots \\ \text{Element } 2 \rightarrow [\ \ ]_{2\times 2} \\ \text{Element } 1 \rightarrow [\ \ ]_{2\times 2} \end{array} \right.$$

CDEEP
IIT Bombay
EE 725 L16 /Slide 3

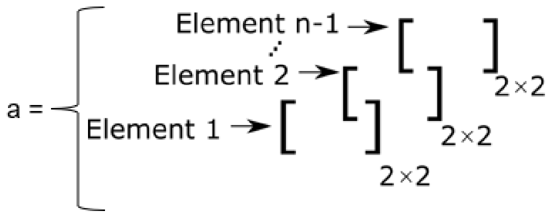
Electrical Equipment and Machines: Finite Element Analysis
element 2: $X_n(1) = 1/10, X_n(2) = 2/10$
Prof. S. V. Kulkarni, EE Dept., IIT Bombay

We have completed the discretization part of the code which is the first step. Now the second step is the formation of local coefficient matrices. In the previous lecture also, we have seen that the local coefficient of matrices of every element are calculated individually. Since there are only 3 matrices (for 3 elements), we wrote them side by side on a single slide.

Let us consider that we have 100 elements in a problem domain, then we will have to calculate 100 local coefficient matrices. That means what? From the coding point of view, you need to define 100 variables to store those 100 matrices. This is a very difficult task and you cannot develop a generalized code also. If we want to use 1000 matrices, then we have to change the complete code for 1000 variables, which is a very difficult task.

Generally we know about the matrices with rows and columns, with only rows, and with only columns. Along with this, in both Scilab and MATLAB, we can define 3-dimensional matrices. That means we can define n number of 2×2 matrices and we can stack them one after the other in a single variable.

That is how we can store all the element coefficient matrices in a single variable. So that even though if you change the number of elements, then the number of variables in the code will not change. And this logic is same in the case of 1D and 2D problems. So, in the FEM code for 2-dimensional problems also, we can see the same 3-dimensional matrices. The 3-dimensional matrix will be as shown in the following figure.



The local coefficient matrix of element 1 will be stored in the first matrix like the way shown in the above figure, after that element 2 matrix will be stored and then element 3 matrix will be stored, like this all the matrices will be stacked in a single variable.

Going further we will see how these element coefficient matrices are formed and how they are stored. The formula for element coefficient matrix that we have seen in the previous lecture is given below.

$$a = \frac{1}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

By looking at the above expression, we have to first identify all the common things in it. Identification of such things will make your code simpler. So, in the above expression, what are common? The diagonal entries of matrix are $\frac{1}{l}$ and its off-diagonal entries are $-\frac{1}{l}$. In the present problem, the complete matrix is constant, only thing you need to calculate is l for each segment. Since we have chosen equispaced discretization, this l is also constant.

But if you change the discretization to non-equispaced, then l will be a variable. So, to make the code more general, we have used l as a variable defined with the following command.

```
L=abs(xn(2)-xn(1)); // length of each element
```

This length will be calculated automatically for every element. Now we need to get the geometrical details of the element under consideration. Here, geometrical details are nothing but the coordinates of each node.

We have to get x coordinates of the 2 nodes of each element. For the first element, local node number 1 is global node 1 and local node number 2 is global node 2. This we have seen in the explanation for t matrix. The first row of t matrix indicates the local node number 1 and the second row indicates the local node number 2.

Here, t(1, element) will be 1 and t(2, element) will be 2 for the first element. And in the following part of the code x(1) is nothing but 0, x(2) will be 0.1 for element 1. The following lines of the code will give the coordinates of the nodes for different elements.

```
xn(1)=x(t(1,element));
xn(2)=x(t(2,element));
// for element 1: Xn(1) = 0, Xn(2) = 1/10,
//element 2: Xn(1) = 1/10, Xn(2) = 2/10, and so on
```

From this part of the code, we are getting the coordinates of each node of an element using the connectivity matrix (t) information. The t matrix information will give the global node number of each local node. The above part of the code will give the coordinates and then we are calculating the length of each element using L = abs(xn(2)-xn(1)) and abs gives the absolute value of the difference.

For each element of this problem, the coordinate of the second node is greater than the first node. The difference of the coordinates will give a positive value only. If someone start naming the nodes in elements from higher coordinate to lower coordinate, like if we choose starting node as 2 and ending node as 1, then the difference of the coordinates will be negative. To avoid this problem, we have used abs.

The difference will give always a positive value, because length should be a positive number. Now, we calculated the length of each element. Only thing left out is to calculate element coefficient matrix. One more basic thing that we want to highlight here is, if you want to form a matrix with rows and columns then we need to use 2 for loops. One for loop is for rows, second for loop is for columns. Also, if you want to form a row vector or column vector then you need only 1 for loop.

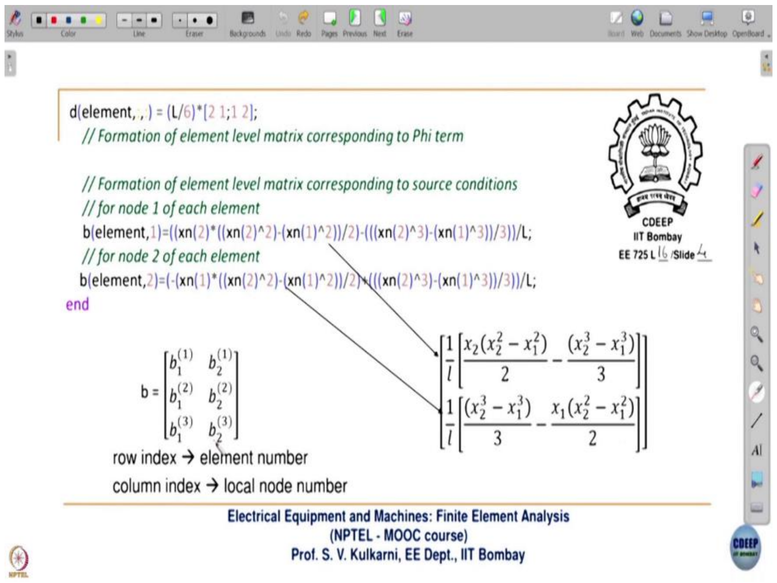
Since this element coefficient matrix is a 2-dimensional matrix, we are using 2 for loops which are given below.

```
for i=1:2
  for j=1:2
     if(i==j) then // Diagonal element of 'a' matrix
      a(element,i,j)=1/L;
     else // Off-diagonal element of 'a' matrix
      a(element,i,j)=-1/L;
     end
   end
 end
```

The i in the first for loop indicates rows and j in the second for loop indicates columns. If $i = j$ then, a(element,i,j) is $\frac{1}{l}$, else it is equal to $-\frac{1}{l}$. Here we want to highlight the first difference between Scilab and MATLAB.

In MATLAB, 'then' in the if statement is not required. But in Scilab, we need to add 'then'. This is one of the differences that we want to highlight. In the present code, this is the only difference. When you use this code in MATLAB, you have to delete this 'then'. So with this we have formed the element coefficient matrices in a single variable, which is a 3-dimensional matrix with all the element coefficient matrices are stacked one after the other.

(Refer Slide Time: 18:07)



```
d(element,:,:) = (L/6)*[2 1;1 2];
  // Formation of element level matrix corresponding to Phi term

  // Formation of element level matrix corresponding to source conditions
  // for node 1 of each element
  b(element,1)=((xn(2)*((xn(2)^2)-(xn(1)^2))/2)-(((xn(2)^3)-(xn(1)^3))/3))/L;
  // for node 2 of each element
  b(element,2)=(-(xn(1)*((xn(2)^2)-(xn(1)^2))/2)+(((xn(2)^3)-(xn(1)^3))/3))/L;
end
```

$$b = \begin{bmatrix} b_1^{(1)} & b_2^{(1)} \\ b_1^{(2)} & b_2^{(2)} \\ b_1^{(3)} & b_2^{(3)} \end{bmatrix}$$

row index → element number

column index → local node number

$$\begin{bmatrix} \frac{1}{l}\left[\frac{x_2(x_2^2 - x_1^2)}{2} - \frac{(x_2^3 - x_1^3)}{3}\right] \\ \frac{1}{l}\left[\frac{(x_2^3 - x_1^3)}{3} - \frac{x_1(x_2^2 - x_1^2)}{2}\right] \end{bmatrix}$$

CDEEP
IIT Bombay
EE 725 L 16 /Slide 4

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

2×2 matrices with constant entries can be entered directly in a matrix form as given below.

```
d(element,:,:) = (L/6)*[2 1;1 2];
  // Formation of element level matrix corresponding to Phi term
```

In this representation, if you enter the numbers in a row one after the other, then the entries will go in a row. To change the row number, you have to keep a semicolon and that will go to the second row. If the matrix is simple with constant entries then you can use this way.

In the entire FEM procedure, either it is 2D or 1D, the D matrix will be a constant matrix with constant numbers. That is why in both 2D and 1D FEM formulations, we will enter the element level D matrices using this representation. So, we have completed the formation of element level

D matrix. Now we will form the element level source matrix. As as we have seen in the previous lecture, expressions for the entries of the element level b-matrix are given below .

$$b^{(e)} = \begin{bmatrix} \frac{1}{l}\left[\frac{x_2(x_2^2 - x_1^2)}{2} - \frac{(x_2^3 - x_1^3)}{3}\right] \\ \frac{1}{l}\left[\frac{(x_2^3 - x_1^3)}{3} - \frac{x_1(x_2^2 - x_1^2)}{2}\right] \end{bmatrix}$$

The entry in the first row of the above matrix corresponds to local node number 1 and the entry in the second row corresponds to local node number 2. Here we can see that these expressions are little complicated and they are different for different nodes.

That is why we are entering these entries separately one after the other as given in the following lines of the code.

```
b(element,1)=((xn(2)*((xn(2)^2)-(xn(1)^2))/2)-(((xn(2)^3)-(xn(1)^3))/3))/L;
//for node 2 of each element
b(element,2)=(-(xn(1)*((xn(2)^2)-(xn(1)^2))/2)+(((xn(2)^3)-(xn(1)^3))/3))/L;
```

Here, we want to highlight that if your functions are common for all the nodes, then you can use for loops to form the matrix. Otherwise, you have to enter one after the other like the way shown in the above lines of the code. This is the third way of entering and it will be used if you have a complicated function in x and y.

As we have discussed earlier, the element level *b* matrix is a 2×1 or a 1×2 matrix as shown in the above expresssion. To store all these element level source matrices in a single variable, we can use a rectangular matrix. Here we are using a rectangular matrix which is given below.

$$b = \begin{bmatrix} b_1^{(1)} & b_2^{(1)} \\ b_1^{(2)} & b_2^{(2)} \\ b_1^{(3)} & b_2^{(3)} \\ & . \\ & . \end{bmatrix}$$

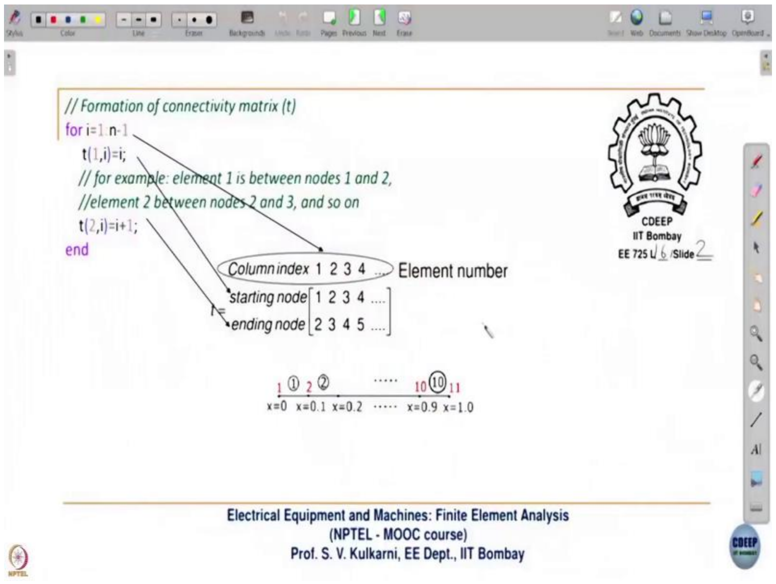
In each row of the above matrix, we are entering the element level *b* matrices. The first row of this matrix corresponds to the first element and the row index corresponds to element number. When element number is 1 that will correspond to the first row and the local node number corresponds to the column number.

The element number 2 will correspond to the second row of the above matrix and the local node number 1 entry will be in the first column. Like this, we will completely enter the information of element level b-matrices in a single matrix (or variable) which is a rectangular matrix.

Here the first row corresponds to the first element and the element index is indicated in the superscript of matrix entries in the above expression, the second row corresponds to the second element, the third row corresponds to the third element. The first column of the matrix corresponds to the first local node and the second row corresponds to the second local node. Like this, we will enter the element level b-matrices.

Till now, we have completed the formation of element level coefficient matrices and we have completed the second step. Now we will go to the third step, which is the formation of global coefficient matrix using corresponding element coefficient matrices. Till now, we have $n-1$ element level coefficient matrices and now we will combine all the element coefficient matrices into a single global coefficient matrix.

(Refer Slide Time: 22:25)



```
// Formation of connectivity matrix (t)
for i=1:n-1
   t(1,i)=i;
  // for example: element 1 is between nodes 1 and 2,
  //element 2 between nodes 2 and 3, and so on
  t(2,i)=i+1;
end
```

$$t=\begin{array}{r|l} \text{Column index} & 1\ 2\ 3\ 4\ \ldots \\ \text{starting node} & \begin{bmatrix} 1 & 2 & 3 & 4 & \ldots \\ 2 & 3 & 4 & 5 & \ldots \end{bmatrix} \\ \text{ending node} & \end{array}$$

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

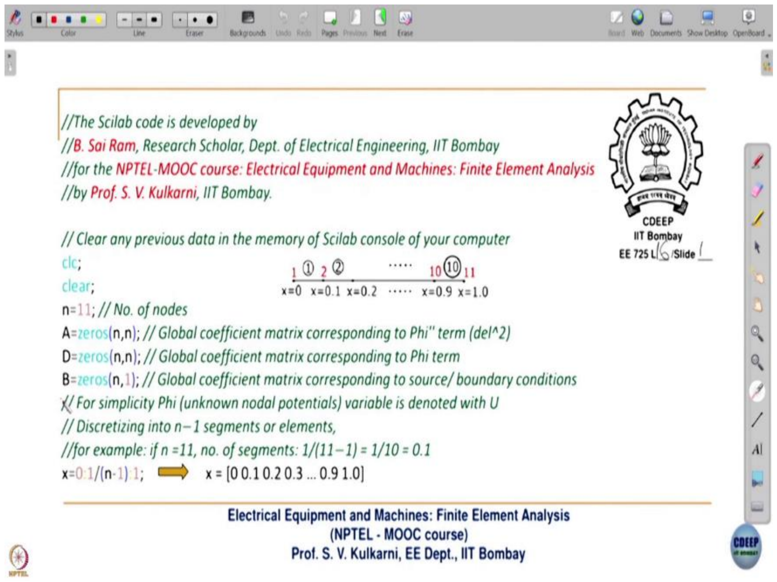
In this slide, the global level matrix is formed from the element coefficient matrices using the connectivity matrix (t). Remember that for each element (column of t matrix), the first row of the connectivity matrix corresponds to the local node number 1 and the second row of the connectivity matrix corresponds to the local node number 2.

As mentioned earlier, we will do element-wise operations in FEM. The for loop in the above slide starts from element 1 to $n - 1$. Then we are taking the global node numbers of each element using the following syntax. The size of nodes defined in the following statement will be 2×1.

```
// global node numbers of each element
nodes=t(1:2,element);
```

The first row of the nodes matrix corresponds to local node number 1 and the second row corresponds to local node number 2. The entries of the nodes matrix for element 1 will be 1, 2. In case of element 2, they will be 2, 3. It means the local node number 1 of element 1 is global node 1, local node number 2 of element 1 is global node 2. For element 1, it may be confusing. We will consider element 2. The local node number of 1 of element 2 is global node 2, local node number 2 of element 2 is global node 3.

(Refer Slide Time: 24:14)



```
//The Scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the NPTEL-MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of Scilab console of your computer
clc;
clear;
n=11; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to source/ boundary conditions
// For simplicity Phi (unknown nodal potentials) variable is denoted with U
// Discretizing into n−1 segments or elements,
//for example: if n =11, no. of segments: 1/(11−1) = 1/10 = 0.1
x=0:1/(n-1):1;
```

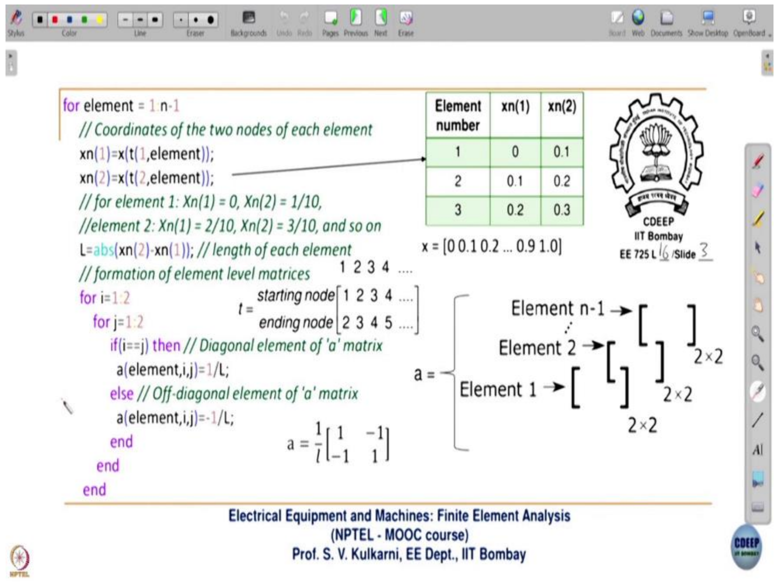
x = [0 0.1 0.2 0.3 ... 0.9 1.0]

CDEEP
IIT Bombay
EE 725 L16/Slide 1

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

If you see in the above slide (the very first slide), we have defined 3 matrices A, D, and B. Now we have to update these three matrices with the entries of element coefficient matrices that we have already formed.

(Refer Slide Time: 24:29)



```
for element = 1:n-1
  // Coordinates of the two nodes of each element
  xn(1)=x(t(1,element));
  xn(2)=x(t(2,element));
  // for element 1: Xn(1) = 0, Xn(2) = 1/10,
  //element 2: Xn(1) = 2/10, Xn(2) = 3/10, and so on
  L=abs(xn(2)-xn(1)); // length of each element
  // formation of element level matrices
  for i=1:2
    for j=1:2
      if(i==j) then // Diagonal element of 'a' matrix
        a(element,i,j)=1/L;
      else // Off-diagonal element of 'a' matrix
        a(element,i,j)=-1/L;
      end
    end
  end
```

Element number	xn(1)	xn(2)
1	0	0.1
2	0.1	0.2
3	0.2	0.3

x = [0 0.1 0.2 ... 0.9 1.0]

$$t = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \;\dots \\ \textit{starting node} & 1 & 2 & 3 & 4 \;\dots \\ \textit{ending node} & 2 & 3 & 4 & 5 \;\dots \end{array}$$

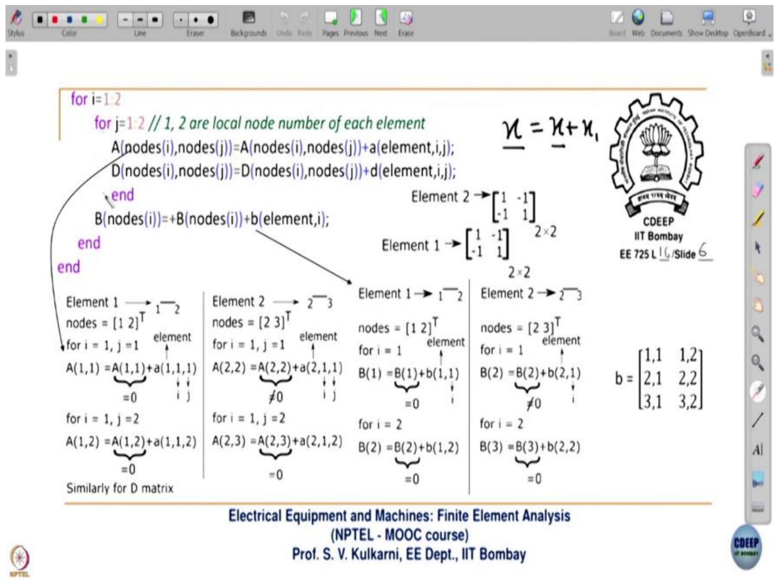
$$a = \frac{1}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

CDEEP
IIT Bombay
EE 725 L16/Slide 3

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

The corresponding global node numbers of each local node are stored in the *nodes* matrix, using this we will take the entries corresponding to each element level matrix and update the corresponding position in the global matrix.

(Refer Slide Time: 24:44)



```
for i=1:2
  for j=1:2 // 1, 2 are local node number of each element
    A(nodes(i),nodes(j))=A(nodes(i),nodes(j))+a(element,i,j);
    D(nodes(i),nodes(j))=D(nodes(i),nodes(j))+d(element,i,j);
  end
  B(nodes(i))=+B(nodes(i))+b(element,i);
 end
end
```

$\underline{x} = \underline{x} + x_1$

Element 2 $\rightarrow \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ 2×2

Element 1 $\rightarrow \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ 2×2

Element 1 $\longrightarrow$ 1—2
nodes = $[1\ 2]^T$
for i = 1, j = 1
$A(1,1) = \underbrace{A(1,1)}_{=0} + a(\overset{\text{element}}{1},\underset{i}{1},\underset{j}{1})$
for i = 1, j = 2
$A(1,2) = \underbrace{A(1,2)}_{=0} + a(1,1,2)$
Similarly for D matrix

Element 2 $\longrightarrow$ 2—3
nodes = $[2\ 3]^T$
for i = 1, j = 1
$A(2,2) = \underbrace{A(2,2)}_{\neq 0} + a(\overset{\text{element}}{2},\underset{i}{1},\underset{j}{1})$
for i = 1, j = 2
$A(2,3) = \underbrace{A(2,3)}_{=0} + a(2,1,2)$

Element 1 $\rightarrow$ 1—2
nodes = $[1\ 2]^T$
for i = 1
$B(1) = \underbrace{B(1)}_{=0} + b(\overset{\text{element}}{1},\underset{i}{1})$
for i = 2
$B(2) = \underbrace{B(2)}_{=0} + b(1,2)$

Element 2 $\rightarrow$ 2—3
nodes = $[2\ 3]^T$
for i = 1
$B(2) = \underbrace{B(2)}_{\neq 0} + b(\overset{\text{element}}{2},\underset{i}{1})$
for i = 2
$B(3) = \underbrace{B(3)}_{=0} + b(2,2)$

$$b = \begin{bmatrix} 1,1 & 1,2 \\ 2,1 & 2,2 \\ 3,1 & 3,2 \end{bmatrix}$$

CDEEP
IIT Bombay
EE 725 L16/Slide 6

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

Since the element coefficient matrices are 2-dimensional, we are using 2 for loops which are shown in the above slide. In these two for loops, we are running from i = 1 to 2 and j = 1 to 2. If it is a 2-dimensional FE analysis which uses triangular elements with 3 nodes, then the for loops will run from i = 1 to 3 and j = 1 to 3. This we will see in later lectures.

Now, we will see the formation of global coefficient matrix for the A-matrix. A(nodes(i), nodes(j)) is getting updated using the following syntax

```
A(nodes(i),nodes(j))=A(nodes(i),nodes(j))+a(element,i,j);
```

A variable will be updated, if we are adding it with some other number. For example, $x = x + x_1$ will update x with x_1.

In the above syntax, A(nodes(i), nodes(j)) is present on both sides. In this we are adding A(nodes(i), nodes(j)) with a(element, i,j) and we are assigning the result to A(nodes(i), nodes(j)). This means we are updating A(nodes(i), nodes(j)). This statement means that we are updating the nodes(i), nodes(j) in global matrix with a(element, i,j). This operation will be explained here.

For element 2, local node number 1 is global node 2 and local node number 2 is global node 3. The entires of nodes that correspond to element 2 will be updated as shown in the following figure.

$$\text{Element 2} \longrightarrow 2\overline{\quad}3$$

$$\text{nodes} = [2\ 3]^T$$

$$\text{for } i = 1,\ j = 1$$

$$A(2,2) = \underbrace{A(2,2)}_{\neq 0} + a(\overset{\overset{\text{element}}{\uparrow}}{2},\underset{\underset{i}{\downarrow}}{1},\underset{\underset{j}{\downarrow}}{1})$$

$$\text{for } i = 1,\ j = 2$$

$$A(2,3) = \underbrace{A(2,3)}_{=0} + a(2,1,2)$$

So, nodes matrix is equal to 2 and 3. The first step that we will be running for element 2 is for i = 1 and j = 1, A(2,2) that means A(nodes(1),nodes(2)). Here, nodes(i=1) is 2 and nodes(j=1) is 2. So, (2,2) position of global coefficient matrix A (A(2,2)) is getting updated with the entry in (1,1) position of element level coefficient matrix of the second element (a(2,1,1)) as given in the following syntax.

$$A(2,2) = \underbrace{A(2,2)}_{\neq 0} + a(\overset{\overset{\text{element}}{\uparrow}}{2},\underset{\underset{i}{\downarrow}}{1},\underset{\underset{j}{\downarrow}}{1})$$

In the above figure, we have written $A(2,2) \neq 0$, because this position of A matrix got updated in element 1.

Here, we are updating the positions in the global coefficient matrix with the corresponding local matrix entries. Now, for i = 1 and j = 2, nodes(i) is 2, nodes(j) is 3. So, for this (2,3) position of A matrix is getting updated with (1,2) position of element level matrix of second element. Similarly for i = 2 and j = 3.

Like this, we will be updating the A matrix. A similar procedure will be followed for D matrix as well. This part will be there in every code to convert local coefficient matrices into the global coefficient matrix. Only thing is, here the numbers in for loops will change as 1 to 3. So, using this code we have formed A and D matrices from the corresponding local coefficient matrices.

Now we will see how to form B matrix (source matrix). The element level b matrix will be like the one shown below.

$$b = \begin{bmatrix} 1,1 & 1,2 \\ 2,1 & 2,2 \\ 3,1 & 3,2 \end{bmatrix}$$

For element 2, global nodes will be 2, 3. For i = 1, we are at local node 1. So local node 1 of element level B matrix will be $b(2,1)$. Here 2 (row number of b matrix) represents the element number. Global B matrix will be updated as given by the following equation

for i = 1

$$B(2) = \underbrace{B(2)}_{\neq 0} + b(\overset{\text{element}}{\overset{\uparrow}{2}}, \underset{\text{i}}{\underset{\downarrow}{1}})$$

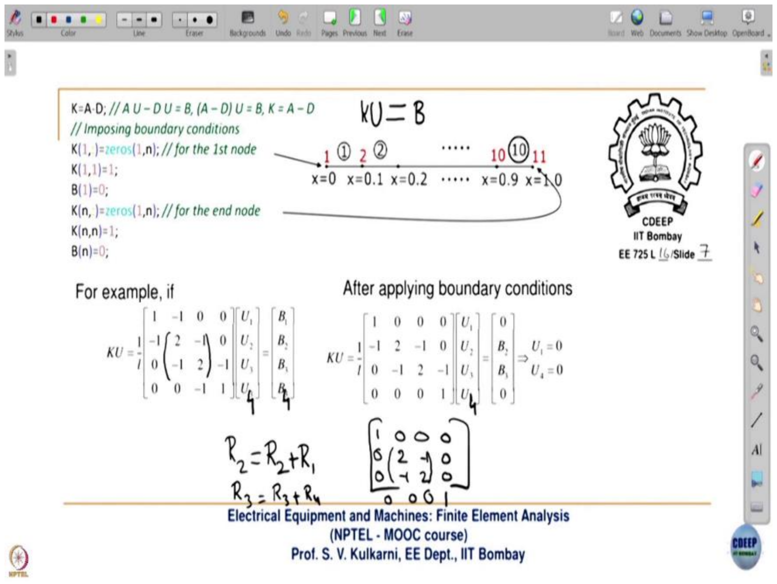
In the above operation, the entry that corresponds to local node 1 of element 2 will be added to global entry 2. Here you can observe that the complexity has reduced by 1. Because source matrix is a 1-dimensional matrix. That is why here we have only one for loop. Similarly for for i = 2, the global node number is 3. The third position in the global matrix is getting updated with the entry of second local node of element 2.

The third position of global matrix B is getting updated with the second entry local coefficient matrix of element 2 ($b(2,2)$). This is how we will convert the source matrix. The following 3 steps inside these for loops will convert our element level coefficient matrices into global coefficient matrices.

```
A(nodes(i),nodes(j))=A(nodes(i),nodes(j))+a(element,i,j);
D(nodes(i),nodes(j))=D(nodes(i),nodes(j))+d(element,i,j);

B(nodes(i))=+B(nodes(i))+b(element,i);
```

Then, once we formed the global coefficient matrices, the next step is simplifying these matrices and solving the final matrix equation.

(Refer Slide Time: 31:27)



```
K=A-D; // A U – D U = B, (A – D) U = B, K = A – D
// Imposing boundary conditions
K(1,:)=zeros(1,n); // for the 1st node
K(1,1)=1;
B(1)=0;
K(n,:)=zeros(1,n); // for the end node
K(n,n)=1;
B(n)=0;
```

$KU = B$

For example, if

$$KU=\frac{1}{l}\begin{bmatrix}1&-1&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&-1&1\end{bmatrix}\begin{bmatrix}U_1\\U_2\\U_3\\U_4\end{bmatrix}=\begin{bmatrix}B_1\\B_2\\B_3\\B_4\end{bmatrix}$$

After applying boundary conditions

$$KU=\frac{1}{l}\begin{bmatrix}1&0&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&0&1\end{bmatrix}\begin{bmatrix}U_1\\U_2\\U_3\\U_4\end{bmatrix}=\begin{bmatrix}0\\B_2\\B_3\\0\end{bmatrix}\Rightarrow\begin{matrix}U_1=0\\U_4=0\end{matrix}$$

$R_2 = R_2 + R_1$

$R_3 = R_3 + R_4$

$$\begin{bmatrix}1&0&0&0\\0&2&-1&0\\0&-1&2&0\\0&0&0&1\end{bmatrix}$$

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

Till now we calculated the global A, D, and B matrices. In the previous lecture we have reduced $AU - DU = B$ to $(A - D)U = B$ by taking U as common and considering $K = A - D$. So we will be solving $KU = B$, after imposing boundary conditions.

In the previous lecture, by imposing boundary conditions we have reduced the dimension of matrices from 4×4 to 2×2. Also, those boundary nodes are the first node and the last node. But in case of complicated geometries and systems, we do not even know the node numbers for which the boundary conditions need to be applied. Also, it will be very difficult to do such simplifications to the matrices. In that case, we will make all the entries as 0 for the row that corresponds to a boundary node and then we will make its diagonal entry as 1. This operation will impose the boundary conditions. For the first and n^{th} nodes, we are applying boundary conditions using the following lines of the code.

```
// Imposing boundary conditions
K(1,:)=zeros(1,n); // for the 1st node
K(1,1)=1;
B(1)=0;
K(n,:)=zeros(1,n); // for the end node
K(n,n)=1;
B(n)=0;
```

In the above lines, for the first row, we are making all the entries as 0 using K(1,:) = zeros(1,n). Then we are assigning 1 to its diagonal entry using K(1,1) = 1. As boundary condition is $U = 0$, we are updating the B matrix entry of the first node as 0. Similarly for nth row that corresponds to the nth node, we are making the nth row completely as 0 and we are assigning 1 to its diagonal entry. We are imposing the boundary condition value in the B matrix. This can be visualized using the following example.

For example, if

$$KU = \frac{1}{l}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}$$

After applying boundary conditions

$$KU = \frac{1}{l}\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 0 \\ B_2 \\ B_3 \\ 0 \end{bmatrix} \Rightarrow \begin{matrix} U_1 = 0 \\ U_4 = 0 \end{matrix}$$

We have seen the above example in the previous lecture. The 4×4 matrix which is on the left hand side is converted into the following 2×2 matrix.

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

This 2×2 matrix is a part of the complete K matrix. In the matrix equation on the right hand side, the first row is completely assigned as zeros and 1 is assigned to its diagonal entry.

Similarly, for the nth row, we are making the complete row as 0 and 1 is assigned to its diagonal entry. Multiplying the modified K matrix with B matrix will give $U_1 = 0$ and $U_4 = 0$. That means in this modified system of matrices, you have imposed the required boundary conditions $U_1 = 0$ and $U_4 = 0$. You can visualize the correlation between this method with the method of reducing the dimensions of K matrix by performing some matrix operations.

If we perform the operation, row 2 is equal to row 2 plus row 1, then the matrix equation will be modified as.

$$KU = \frac{1}{l}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 0 \\ B_2 \\ B_3 \\ 0 \end{bmatrix}$$

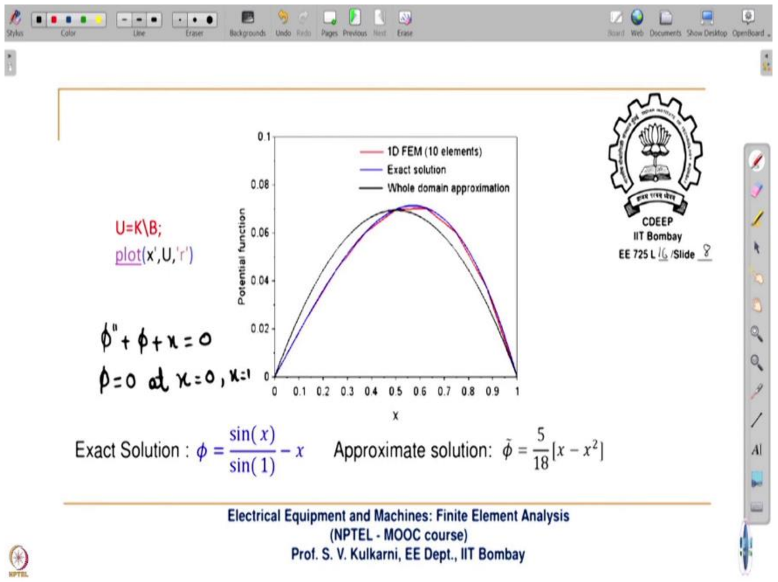
And then making row 3 is equal to row 3 plus row 4, the above matrix is updated as:

$$KU = \frac{1}{l}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 0 \\ B_2 \\ B_3 \\ 0 \end{bmatrix}$$

That means, the second and third rows become independent and this is similar to the matrix equation that we have seen in the previous lecture.

This is how you can visualize similarity between the present way of imposing boundary condition and the one that we have seen in the previous lecture. Also, note that the procedure of making the row corresponding to that node as 0 and assigning 1 to its diagonal entry will be followed in the 2D code as well. Also, since we have only 2 nodes, we have done this operation separately to the two nodes. When we have many nodes, we will use a for loop for this operation also. Now, we have imposed the boundary conditions also and we are ready to take the inverse and solve the matrix equation.

(Refer Slide Time: 36:46)



U=K\B;
plot(x',U,'r')

$\phi'' + \phi + x = 0$

$\phi = 0$ at $x = 0, x = 1$

Exact Solution : $\phi = \frac{\sin(x)}{\sin(1)} - x$ Approximate solution: $\tilde{\phi} = \frac{5}{18}[x - x^2]$

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

We will calculate the nodal potentials by using the command U = K\B. Taking inverse of a matrix by using backstroke is there in all scientific computing software like MATLAB and Scilab. You can directly use U = K\B in both the platforms. Then by using plot(x,U), you will get the variation. The red colour curve which is shown in the figure corresponds to results of 1D FEM code.

Since we have chosen less number of elements (10 elements), there is an error at points between nodes, because in FEM we are minimizing the energy at nodes unlike in the whole domain approach.

Now, in the above figure we have compared the computed results with the exact solution which is in blue color and the black curve is the approximated solution with second-order approximation which we have seen in the previous lecture. As you know these curves are the solutions for the 1-dimensional PDE $\phi'' + \phi + x = 0$, with boundary conditions $\phi = 0$ at $x = 0$ and $x = 1$.

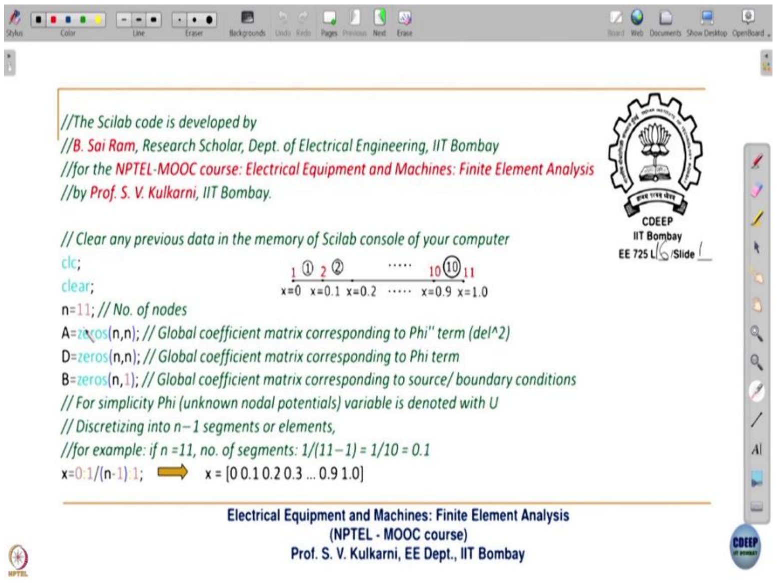
In the figure, you can see that at x = 0 and x =1, the boundary conditions are imposed perfectly. The exact solution is given below.

$$\phi = \frac{sin(x)}{sin(1)} - x$$

This solution and the approximate solution $\tilde{\phi} = \frac{5}{18}[x - x^2]$ with second order approximation, we have seen in the previous lecture. Also, in whole domain approximation to improve the accuracy, we have increased the approximation from second order to third order.

That made us to redo the complete process which involves complicated calculations compared to the calculations for second-order approximation. As we keep on increasing the order, the complexities in the calculations will increase. Also, it is not a general procedure, whereas the code that we have developed is generalized. Here the results for 10 elements are shown.

(Refer Slide Time: 39:21)



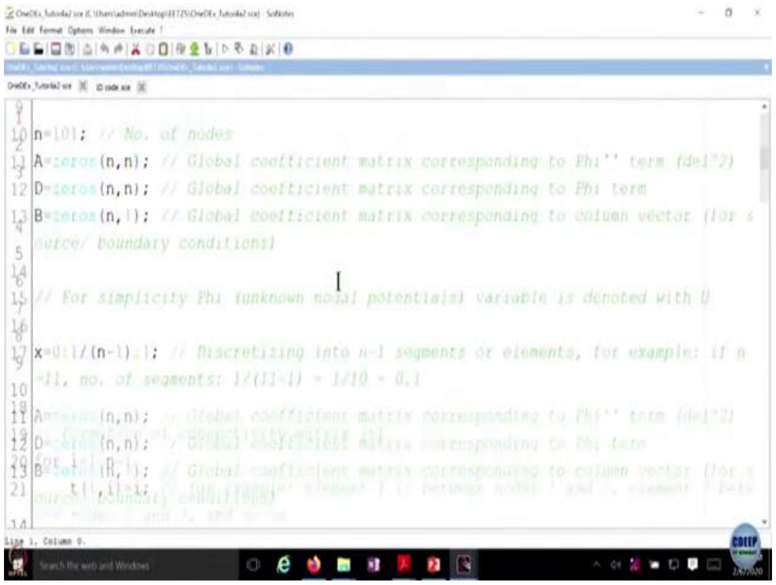
```
//The Scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the NPTEL-MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of Scilab console of your computer
clc;
clear;
n=11; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to source/ boundary conditions
// For simplicity Phi (unknown nodal potentials) variable is denoted with U
// Discretizing into n-1 segments or elements,
//for example: if n =11, no. of segments: 1/(11-1) = 1/10 = 0.1
x=0:1/(n-1):1;
```

1 ① 2 ② ..... 10 ⑩ 11
x=0 x=0.1 x=0.2 ..... x=0.9 x=1.0
x = [0 0.1 0.2 0.3 ... 0.9 1.0]
CDEEP
IIT Bombay
EE 725 L16/Slide 1

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

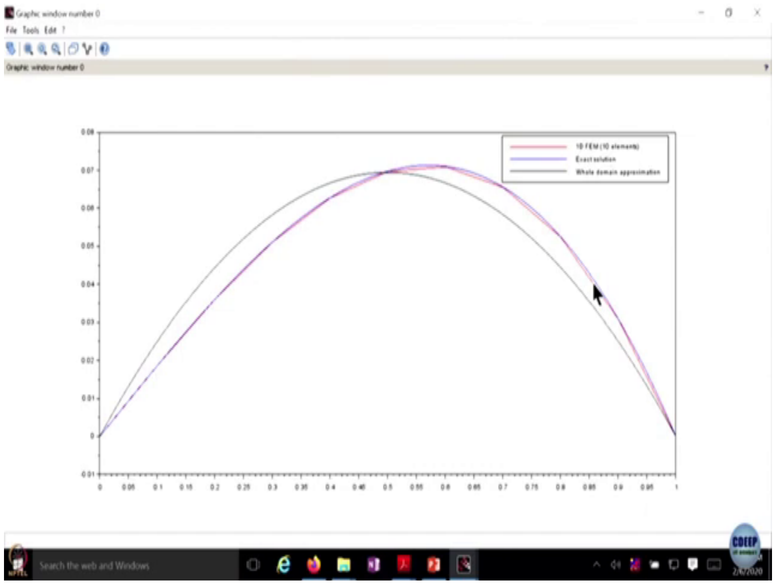
Now in the code, we will change the number of elements by changing the very first line of the code (n = 11). The accuracy of the result will be improved without any further changes in the code. First we will change n = 11 to n = 21 and then n = 31 without touching any steps in the code to improve the accuracy. So what we are trying to say is, this kind of FEM code will be general and only thing you need to change is its discretization. The improvement in accuracy is shown below.

(Refer Slide Time: 39:44)



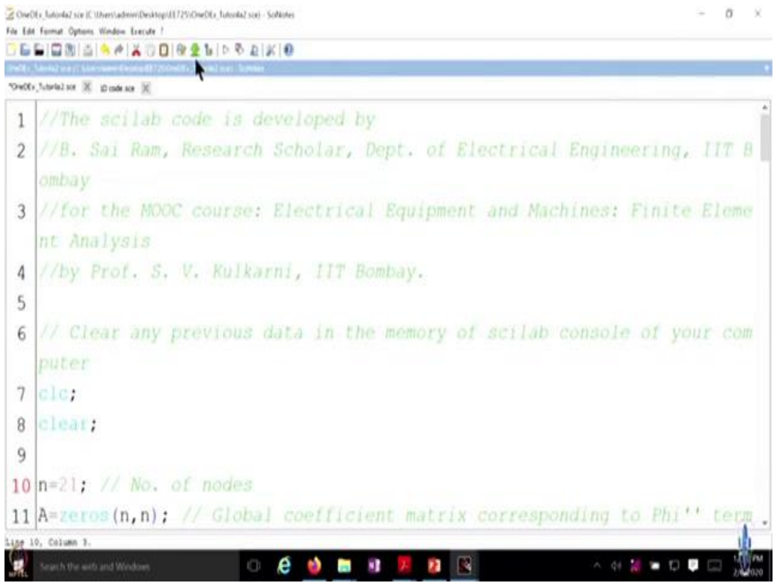
```
n=101; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term (del^2)
D=zeros(n,n); // Global coefficient matrix corresponding to Phi term
B=zeros(n,1); // Global coefficient matrix corresponding to column vector (for source/ boundary conditions)

// For simplicity Phi (unknown nodal potentials) variable is denoted with U

x=0:1/(n-1):1; // Discretizing into n-1 segments or elements, for example: if n=11, no. of segments: 1/(11-1) = 1/10 = 0.1
```

This is the code in Scilab interface which we have seen in the presentation. In this code, we will be changing only the value assigned to variable n. Now running the code for n = 11 will give the result in the following figure.



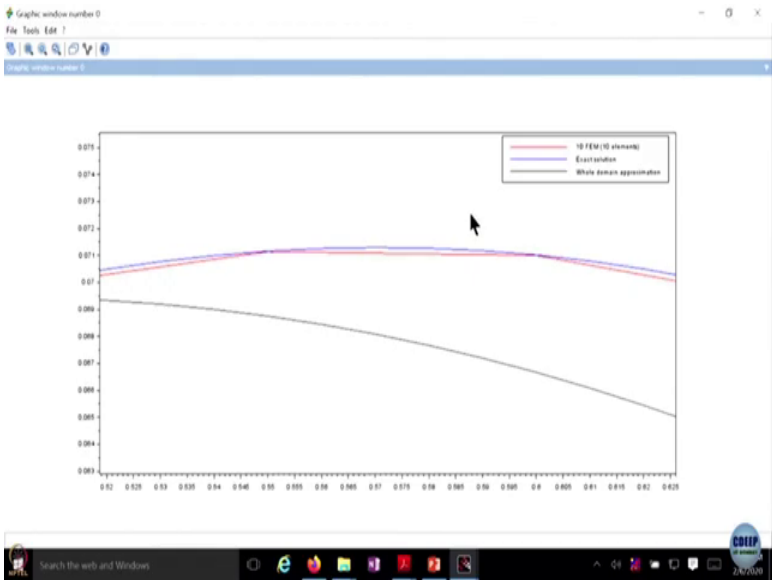
So you can see that solution in red colour is very much approximate and the error is more.

(Refer Slide Time: 40:22)

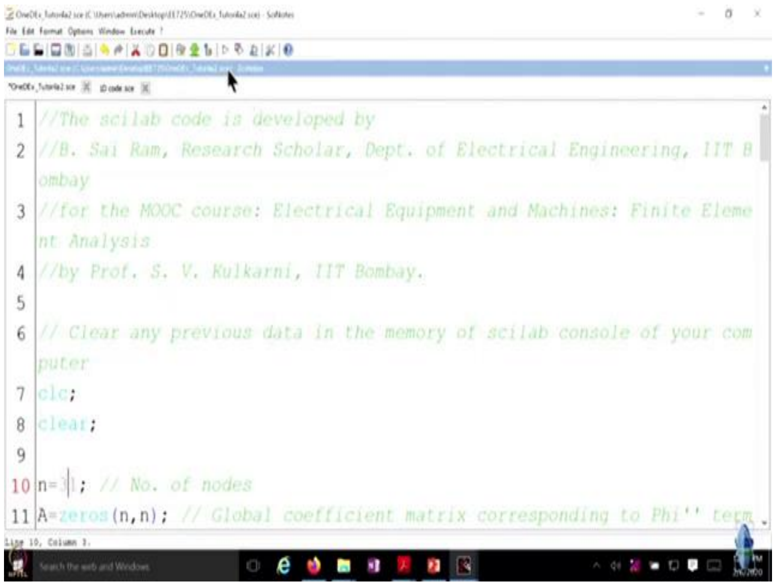


```
//The scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of scilab console of your computer
clc;
clear;

n=21; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term
```

Now changing n = 21, you can see both solutions got overlapped with little error as shown in the following figure.

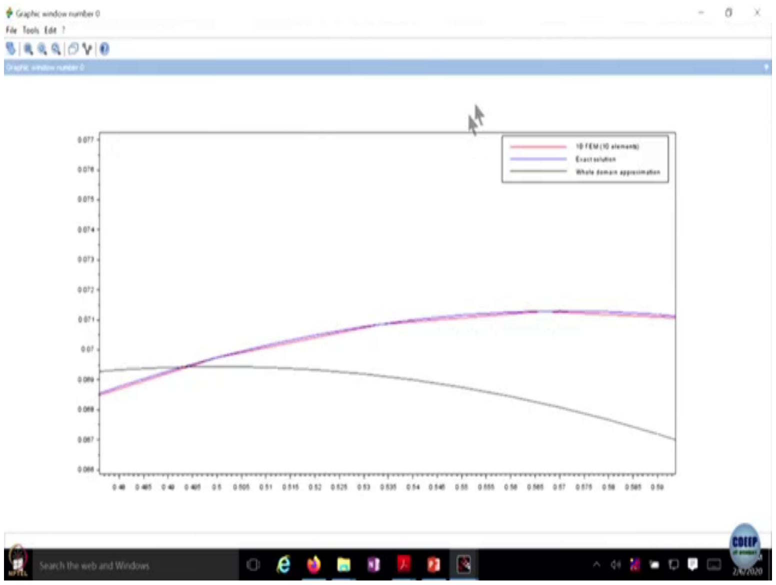


(Refer Slide Time: 40:42)

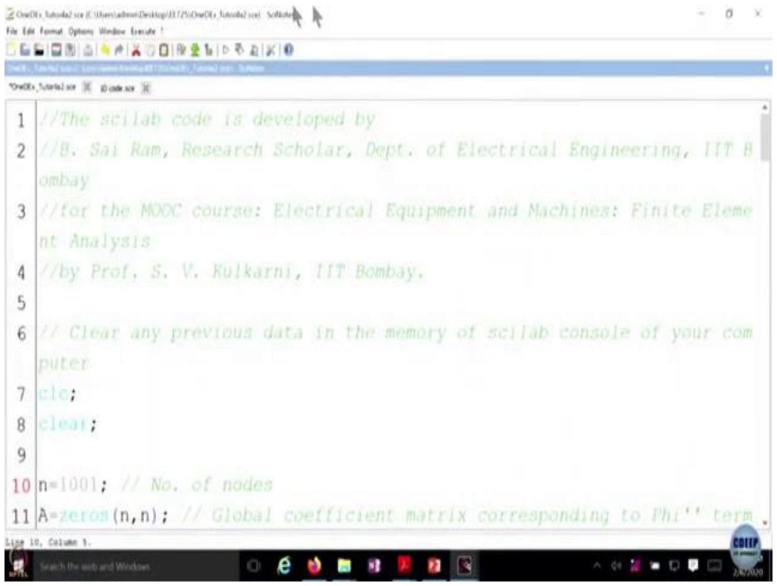


```
//The scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of scilab console of your computer
clc;
clear;

n=31; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term
```

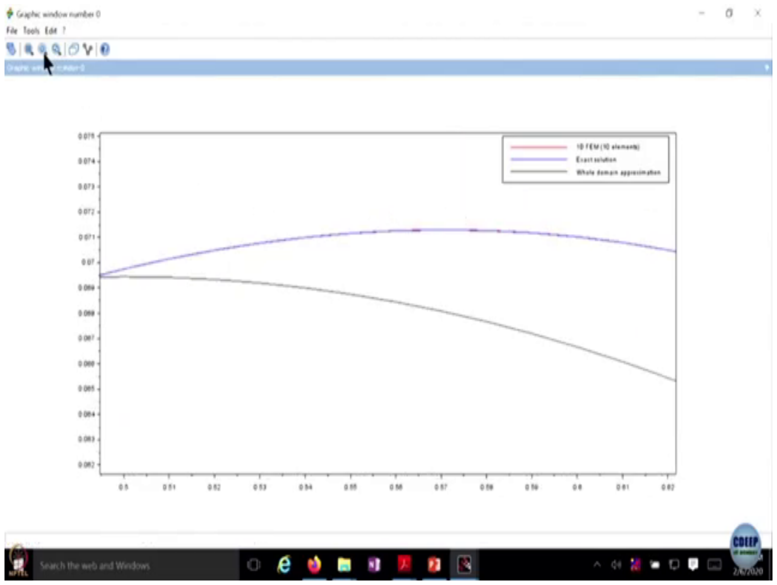
Now we are increasing n to 31 and it got almost overlapped. The complexity in computation is also reduced. We have wrote the code for 11 elements and then we got solutions for different number of nodes. Using the code, we can get the solution for 1000 also.



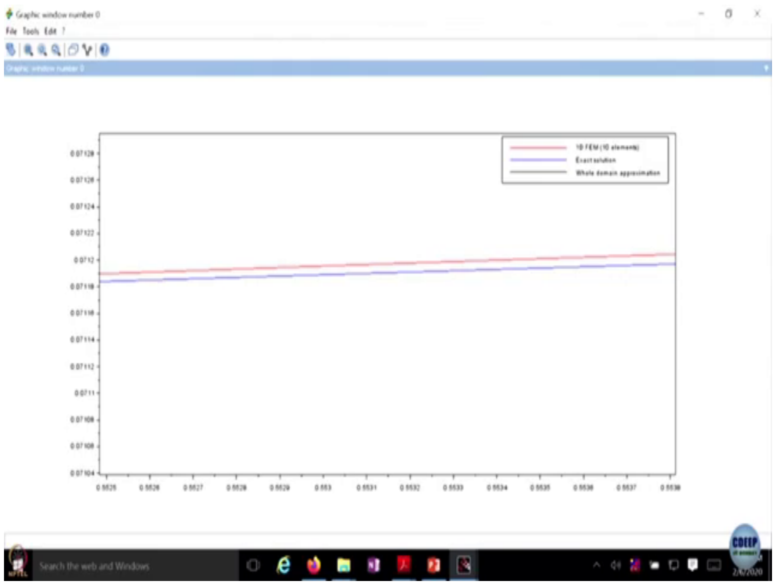
(Refer Slide Time: 41:11)



```
//The scilab code is developed by
//B. Sai Ram, Research Scholar, Dept. of Electrical Engineering, IIT Bombay
//for the MOOC course: Electrical Equipment and Machines: Finite Element Analysis
//by Prof. S. V. Kulkarni, IIT Bombay.

// Clear any previous data in the memory of scilab console of your computer
clc;
clear;

n=1001; // No. of nodes
A=zeros(n,n); // Global coefficient matrix corresponding to Phi'' term
```

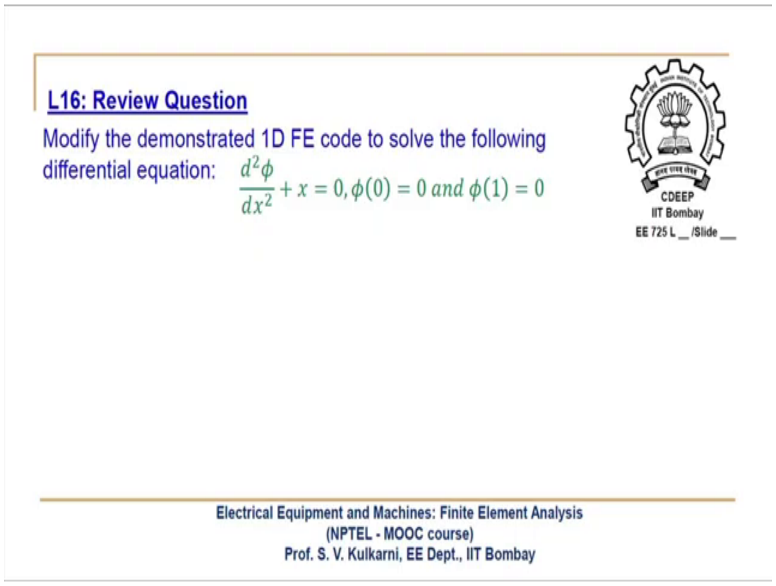


With this in the above figure, the solution got overlapped. In the figure, you cannot see that red line and the blue line is overlapping that red line.



You can see red line if we zoom in the figure too much. So, if you develop a code, you can make it more general and it will help you in improving the accuracy of computations. Thank you.

(Refer Slide Time: 41:49)



L16: Review Question

Modify the demonstrated 1D FE code to solve the following differential equation: $\frac{d^2\phi}{dx^2} + x = 0, \phi(0) = 0 \; and \; \phi(1) = 0$

CDEEP
IIT Bombay
EE 725 L __ /Slide ___

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