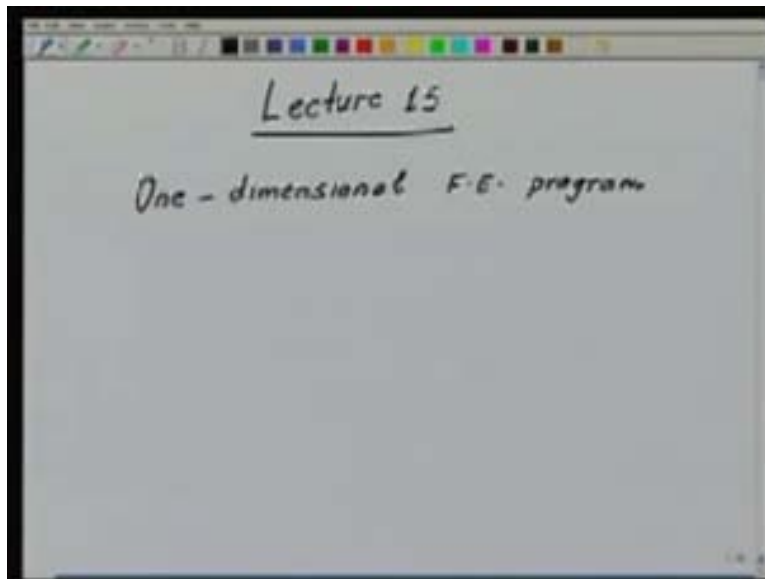


Finite Element Method
Prof. C. S. Upadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur
Module – 5 Lecture - 2

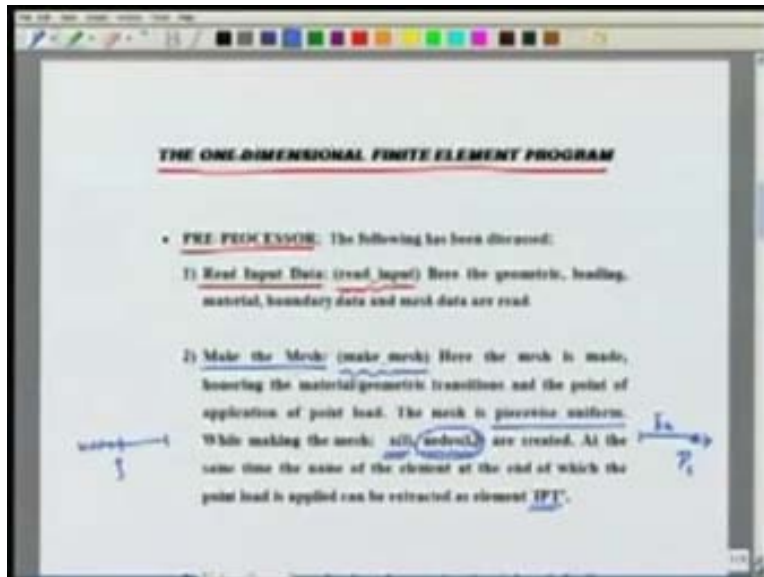
Let us start today's lecture with recap of what we have done in the previous lecture. In the previous lecture, we had started talking about the one dimensional finite element program.

(Refer Slide Time: 00:20)



Here we have outlined (Refer Slide Time: 00:40) what are the various building blocks of finite element program and we had said that it consists of the pre-processor, the processor and the post-processor. In the previous lecture, we had outlined, how to make the mesh. How to read the input data and how to make the mesh; given the kind of features that we wanted to incorporate in a program. In this lecture, we will take that further; we will go beyond the mesh. We will try to see what are the other building blocks we need. Once all these building blocks are there, then we have essentially everything that we need to make one dimensional finite element program. Let us go and see the basic structure of finite element program and the various pieces that we need.

(Refer Slide Time: 01:40)

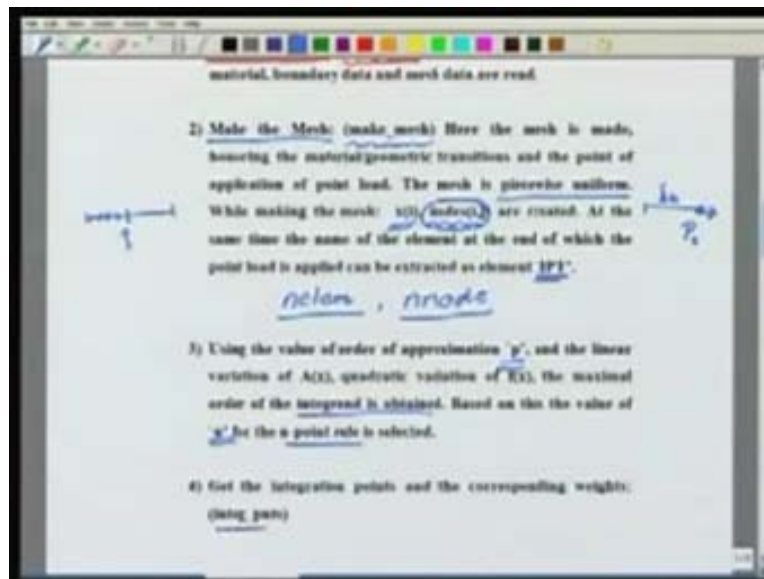


We have come to the one dimensional finite element program that we need to develop. Here as we had said a pre-processor is what we are at; we had already done the part about reading the input data. We had created this routine called Read Input and what we had done here is, we had read in the geometric data, the loading data, the material data and a boundary data and also what kind of meshing we needed, that is how many elements in each piece, what is the order of approximation and so on. These are all the data that were input in the read data program.

We have looked at how to make the mesh. This was as we had given a name done through a sub program or sub routine called Make mesh. The idea is, given the mesh data and the material data and the loading data that is available to us, we would like to make the mesh in a particular way so that it honors the material transitions, the places where the point loads are applied and if need be, we can also have different size meshes in different regions. Here the mesh was divided into piecewise uniform meshes. That is, in each piece, if I have a piece here, the domain in each piece I have a different size element and these are the transitions coming due to the material and the point load. So given the mesh, we made a set of loads and then the global coordinates given by x , then we went and created the elements and we said that for an element, I would like to know, what are the extremities? That is the two nodes at the two ends of the elements.

This data was given in the nodes i, j array; these arrays were created. At the same time, while we were making the mesh, because we know these material transition points and also the point where the point load is applied, we can extract the name of the element at the end of which the point load is applied. If this is the element, here is the point load applied (Refer Slide Time: 04:36). We will extract the name of that element and I will call it IPT. It is not **sacrosine** but we can do it like this. One can give any name that he wishes; I choose to give IPT.

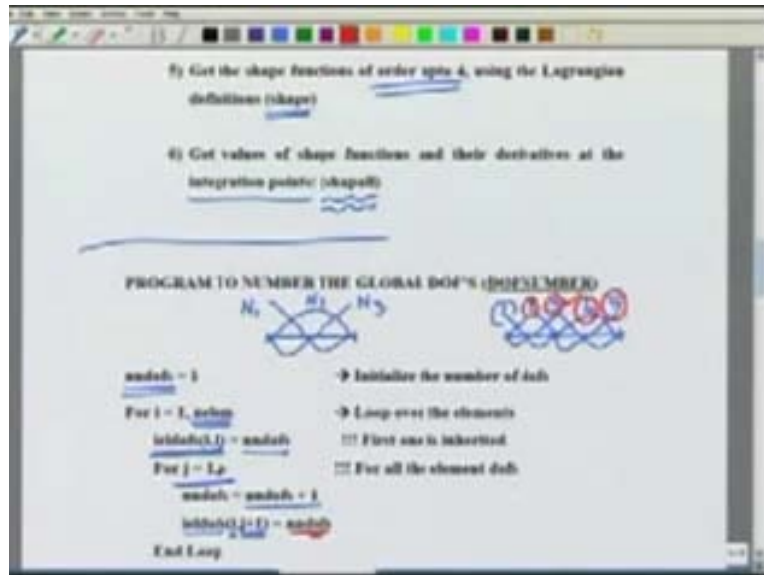
(Refer Slide Time: 04:48)



Once the mesh is ready, then here we had also obtained these parameters $nelem$. $nelem$ is the total number of elements that we have in the mesh. Also we had the total number of nodes in the mesh given by $nnode$. Using the order of approximation P , which is given to us, we had assumed uniform approximation and the linear variation of the material, the cross sectional area given by A_x , quadratic variation F_x , the maximum order of the integrand is obtained. This we have discussed in the detail in an earlier lecture. Based on this, the value of n that is what point rule I have to use in the Gauss quadrature rule; this end is selected.

Given this n , we can get the integration points and the corresponding weights from the routine intake points. These are the things that this routine will do for us - the intake points routine.

(Refer Slide Time: 06:27)



After the intake point routine has been obtained, we had also developed the routine shape as discussed in the previous lecture, where we could output the shape functions and the derivatives using the Lagrange definitions, for order P upto 4. We can get upto the 4th order shape functions; this routine shape would output that. Then we had decided to have another routine which will take these integration points that we had obtained out of intake points and at these integration points, it is going to output the value of the shape functions and the derivatives in the master element and store them away. This was done in the routine called shapall; why was this needed? This was needed for our element calculation because as we are looping over the elements, trying to compute the element stiffness matrix and the load vector, we do not want to call the shape function program at every integration point, get the output of the shape function, use it and then go to the next one. This causes repetitive work and that costs in terms of time.

We had said over the integration points since they are the same for each element, we call the shape functions, store them away and in the elements as we go and do the element calculation, we are simply going to read out those values from the stored set. Then comes this program (Refer Slide Time: 07:56) to number the global degrees of freedom. What we have done till now we have discussed this point. We would like to enumerate the degrees of freedom in the elements. How are we going to enumerate the degrees of freedom? What do we mean by that? Let us say, this is my mesh (Refer Slide Time: 08:19). I have a quadratic approximation. How

many global basis functions we have? We have 1, 2, 3, 4 and 5. I have taken the two element mesh where I have this 5 global basis functions. How do I construct them? We have the subroutine or the function called the dof number where we will go element by element, construct these global basis functions and give a one to one correspondence between these global basis functions and the element level degree of freedom; that is if I am in element 2 for example, this was N_1 , this was N_2 and this was N_3 for element 2. Which global basis functions ϕ does this N_1, N_2, N_3 correspond? This data we are going to construct in this routine. If we see N_1 , the local that is the element 2 is 1 corresponds to global 3, 2 corresponds to 4 and 3 corresponds 5. How do we go about doing it? We have the total number of degrees of freedom in the mesh given by $nndofs$ and we initialize to 1. Then we start loop over the elements. As we loop over the elements, we have this array $i, nndofs$; $i, 1$. i is the element index and the second one which is J , this is the local degree of freedom. That is whether it is 1, 2, 3 for every element we will have P plus 1, such degrees of freedom in the element.

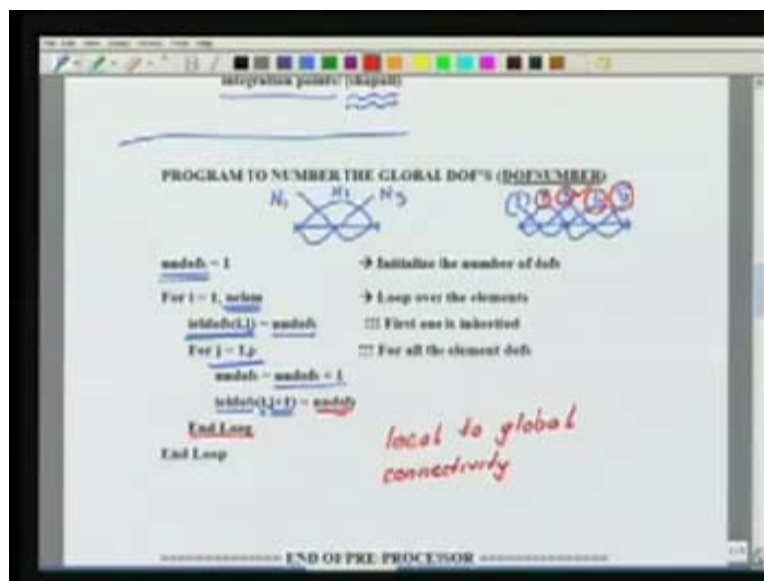
For the element 1, here I said $i, nndofs$ $i, 1$ is equal to $nndofs$; what does it mean? For element 1, the first degree of freedom of the element 1 corresponds to the global 1. Then I go over the remaining P degrees of freedom in the element and I keep on constructing this global degrees of freedom and updating the element to the global connectivity. I am going over looping over this j equal to 1 to P . I increment the $nndofs$ that is the total number of degrees of freedom remembering that I should not repeat this increment; that is, one degree of freedom should not be counted twice. So I increment this degree of freedom count then for this element $i, 1$ is already set. 2 becomes $nndofs$, the new $nndofs$, then 3 becomes new n $nndofs$ and so on and I keep on doing this.

If we see this for 2 element mesh, for element 1 $nndof$ 1 is 1, starting of, so I get for element 1 the first entry is the global 1 which is what we have here. For $nndofs$, I loop over P so P because I have taken 2. So it will be 2. Then I increment $nndofs$, so it becomes 2, then for the element 1, the second one becomes 2. Then I go and increment $nndofs$ again and this becomes 3. For the element 1, the third one becomes 3 then I go to the element 2.

When I come to the element 2, first one of element 2 is 3. Count here has stopped at 3, so the first one of element 2 automatically becomes 3 then the second one of element 2, I come here in

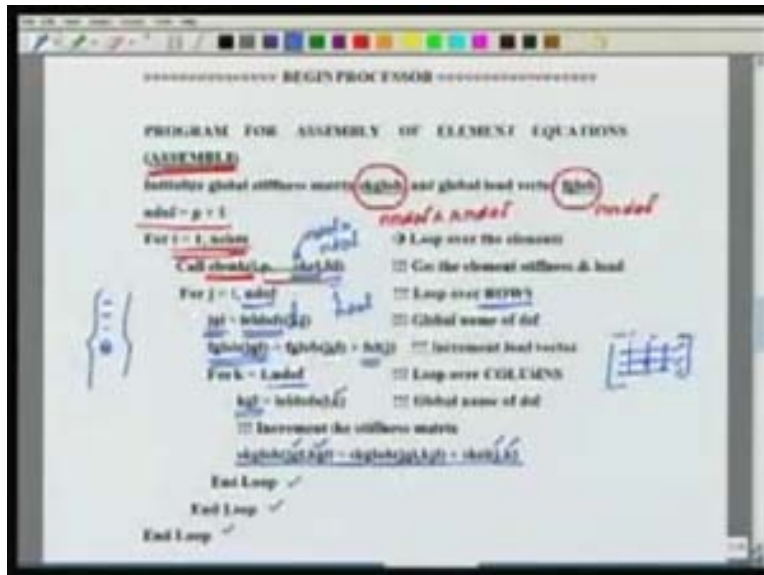
this loop, I add, it becomes 4. So it makes it 4 and the third one it makes it 5 (Refer Slide Time: 12:40). This way for any P, we can keep on constructing the global degrees of freedom and assigning to the element, this connectivity between the element 1, 2, 3 up to P plus 1 to the global i, j, k. So this I am going to do in the loop. Once I have done this then I stop loop over the element degrees of freedom, loop over the elements and I have essentially the one to one correspondence between the local degrees of freedom in the element to the global degrees of freedom in the domain.

(Refer Slide Time: 13:20)



Here I have local to global connectivity. This is very important from the point of view of adding up the element entries to get the global stiffness matrix and load vector matrix entries. With this once we have read in the input data, we have made the mesh, we have obtained the integration points as stored as an additional part, the values of the shape function at this integration points, then constructed the enumeration of the degrees of freedom then our pre-processor has ended. This job has to be done irrespective of what problem I am going to solve; what is the EA? What is the F? That is going to be used next. So the pre-processor ends here.

(Refer Slide Time: 14:36)



We come to our processor. What does the processor really do? Processor is actually the main building block, where I am going to start constructing for the given problem that is, for the material data, loading data, boundary conditions, I am going to construct the global stiffness matrix K and the global load vector F ; apply the boundary conditions and then solve the problem. This is what our processor is going to do. The first part of the processor is the assembly program. Essentially, we will call it by the name Assemble. Again we can call it by any name. The first job is to initialize global matrices. I am calling global stiffness matrix as K_{globe} . One can call it any name and the global load vector, F_{globe} . So K_{globe} is essentially of size n_{dof} by n_{dof} and F_{globe} of size n_{dof} .

In the previous program, as we kept on adding the degrees of freedom, our counter for the total number of degrees of freedom kept getting incremented and at the end of the program, we had the total number of degrees of freedom in the domain. Once we have all this I have the n_{dof} and so on. So we can first initialize the global stiffness and the load vector. In the element, how many degrees of freedom are going to be active? I am given the order of approximation P . It is going to be $P + 1$ approximation, so we name another variable called the n_{dof} which is equal to $P + 1$. Then we start looping over the elements; this is the crucial part of the assemble program. Here as we start looping over the elements, for the element as we had created this k routines,

which was used to give us the element stiffness and the load vector entries, I will call this routine for each of the elements I, as I am looping over the elements.

Giving the input data that is met and getting the output data in the form that we need. So it gives out to me, the element stiffness matrix 'skel' and the element load vector 'fel'. 'skel' is of size ndof by ndof and of the load vector is of size ndof. So it is of size P plus 1 by P plus 1 and this is what we get. Where do we put it, in the global stiffness matrix and the load vector? That is what we need from the assembly point of view. For that we start looping over the degrees of freedom for the element; that is; we loop over j equal to 1 to ndof. As we are looping over the degrees of freedom for the element then for each of these degrees of freedom, I get its global name. So for the element I, now this has become big I, it should have been small i.

For the element i, I take the local j and from what we have done in the previous routine, we are now able to find, what is the global names of the degree of freedom. It is, we call it by 'jgl' that is J global. Then what do we do? We have obtained the element level load vector. We take the corresponding global load vector entry that is given by jgl. So F global entry that is the number here is, if I do this, this is jgl. To this entry, I add the element entry (Refer Slide Time: 18:55). There is one to one correspondence between the j and the jgl through which I add this element entry in the global entry. Once I have done that I go over, this is essentially a loop over the rows. Essentially, I am looping over global rows to that I am adding the local rows in the right position; this is for the load vector.

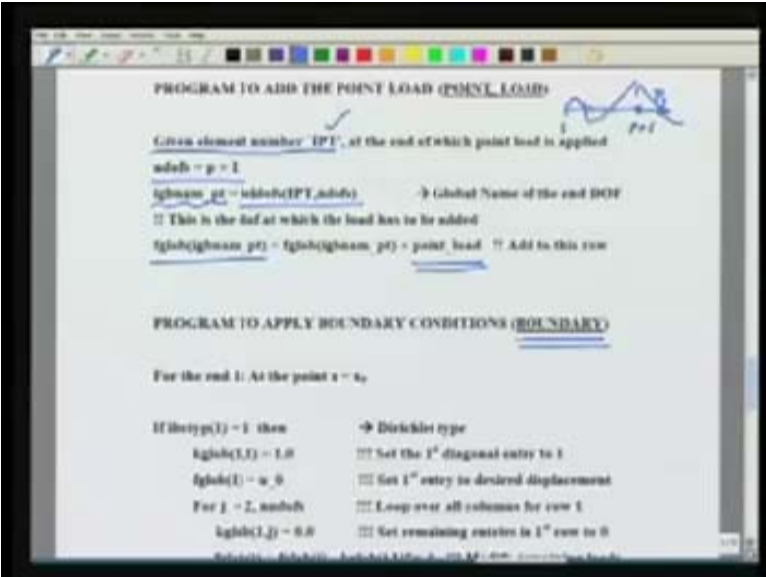
Similarly, I will have to loop over the columns. These are my rows (Refer Slide Time: 19:30) and these are my columns. I start looping over the columns and I call them by the variable k. Again I loop over the number of degrees of freedom which are there in the element. Again I find the global name of the element degree of freedom k. For the element, I find the global name kgl corresponding to k. Once I have the global name of the column and the row corresponding to which this element entry has been obtained, I am going to simply add that entry in the corresponding global row and column. That is what exactly I am doing here. In the sk globe, that is the global stiffness matrix, I go to the global row, the global column to that I add the local row and the local column entry and I am done.

Essentially I end the loop over the columns then I end the loop over the rows and then I end the loop over the integration points. One thing that I try to do in this construction, this is not the full pledged program is that I have done my writing in an indented way. That is, if I have the outer loop which is the loop over the elements, then I will essentially, these two points match, the end loop for the outer loop matches with the for where it starts.

Everything inside this outer loop is indented. Generally, one indented by 3 or 4 empty spaces and then I start this part - writing the inner part. In the inner part, I come to this loop, another embedded loop; this loop ends here (Refer Slide Time: 21:36). This loop is over the rows. Again, whatever is inside this loop is again nicely indented so that we can very clearly see the embedding of the loops inside each other. That is essentially a very important thing from debugging point of view, from writing the program point of view, first of all, it looks very nice and secondly, we will not be mixing up the loops.

The third loop, the inner most loop is over the column which ends first, then the outer loop then the outer most loops over the elements ends. This is something that a student should keep in mind, when writing the program. For this, we needed the element calculations which are coming here. Those element calculations will be provided by our routine element l m k which we have already discussed in the 13th lecture.

(Refer Slide Time: 22:41)



```
PROGRAM TO ADD THE POINT LOAD (PMNL_LOAD)
  Given element number 'IPT', at the end of which point load is applied
  nndofs = p + 1
  iglobn = pt = nndofs(IPT,nndofs)  -> Global Name of the end DOF
  !! This is the dof at which the load has to be added
  iglob(nndofs pt) = iglob(nndofs pt) + point_load  !! Add to this row

PROGRAM TO APPLY BOUNDARY CONDITIONS (BOUNDARY)
  For the row i: At the point x = x_i

  If (ityp(i) = 1) then  -> Dirichlet type
    iglob(i,i) = 1.0  !! Set the 1st diagonal entry to 1
    iglob(i) = u, 0  !! Set 1st entry to desired displacement
    For j = 2, nndofs  !! Loop over all columns for row i
      iglob(i,j) = 0.0  !! Set remaining entries in 1st row to 0
```

After we have assembled in the local element stiffness and the load vector entries into the global stiffness, the global load vector entries, then what are we left with? First, we have to account for the point loads. To account for the point loads, I have a routine called `point_load`; one can give any name. For this routine, from the meshing routine, I have already extracted the name of the element at the second node of which, or the last node of which, the `point_load` is applied. I know the name of this element IPT.

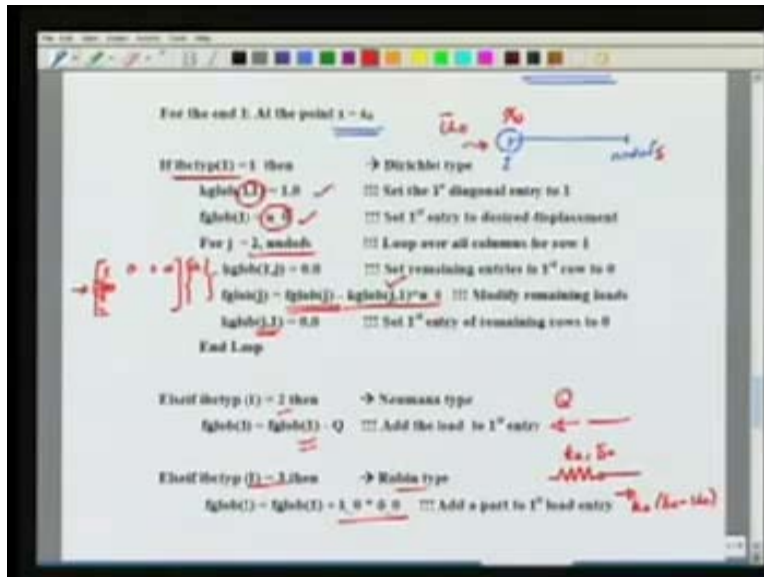
For this element, there are $P + 1$ degrees of freedom. I go to the last degree of freedom because if we remember, this one is the $P + 1$ degree of freedom. This is the first one and we have the others in between. I take the $P + 1$ degrees of freedom; I find its global name. Again using my `ieldof`, we see how crucial this `ieldof` information becomes and this is the kind of data structure that will also be used in 2D and the 3D programs because there life will be a little more difficult. I find the global name of this degree of freedom then the things are very simple. Then I simply go to the corresponding entry in the global load vector and to which I add essentially the point load, the value of the `point_load`. The value in this case we mean that is the positive direction. So `point_load` is acting in the positive direction. If I have to have a negative `point_load`, I should give it as a minus value. It is going to add this value to the corresponding entry; as simple as that.

Why is it going to add to that corresponding entry? Because the only V which is non zero here is this IGB name that is the basis function corresponding to this one.

Once we have this, we have taken care of the point load and here we have decided to add only one point load. An extension of this takes care of multiple point loads, so it is also not difficult. We have to find essentially the locations of the point of application of the loads. Find the corresponding elements at the extremity of where it has been applied and then simply go and find the global degree of freedom corresponding to this extremity and add the value of the `point_load` that corresponding load vector entry. This is how we will take care of point loads.

We have not yet handled the boundary conditions. We have to apply the boundary conditions to our problem; we have a routine called `boundary`. This routine essentially will do the job of applying the boundary conditions and this part has to be done very carefully because if this is wrong, then my answers are going to be wrong.

(Refer Slide Time: 26:15)



How does the boundary condition routine function? What do we have in the domain? We have the two extremities where the boundary conditions have to be applied. The first extremity is the point 1, is the degree of freedom 1. The next extremity is the degree of freedom $nndof$. So this is where the action has to be, at these two extremities (Refer Slide Time: 26:40). Let this extremity correspond to point x_0 that is the first point. We go to this point x_0 and here we would like to first apply the boundary condition that has been specified for the point x_0 . As we have done in the input data, tell me what kind of boundary condition you want to specify at this end? That is, given by this flag ibc type 1. If it is equal to 1 then it is a Dirichlet type boundary condition; that is, I am going to fix the value of the displacement at that point. If it is type 2, then it is a Neumann type of boundary condition; that is, I am going to fix the value of the load at this point; that is, I am applying a point load there. If it is of type 3 then it is mixed or Robin type of boundary condition. Here (Refer Slide Time: 27:48) if we have an end spring which applies a load which is a function of the end deflection itself; this is what we mean by the mixed condition.

Here, how are we going to do these things? Take care of these various types of boundary conditions that can be applied in this problem. Let us see first the Dirichlet type. For the Dirichlet type, I am going to specify the value of the deformation here and this is given u_0 . The displacement at this end, this is given by the value u_0 . u_0 need not be 0 always. I am giving the

value of u_0 and I want to enforce this in our calculations in such a way that indeed, the displacement at this point 1 comes out to be u_0 .

How do I do it? In the global stiffness, I go and take the first diagonal term, 1, 1 and I set it to a value of 1 (Refer Slide Time: 28:52). I have the global stiffness; I take the first diagonal term; set it to a value 1 then correspondingly, first load term I am going to set it. I have calculated something but now I am going to overwrite it and set it to the given value of the displacement u_0 . u_0 is already known; so it has to be eliminated from the rest of the equation.

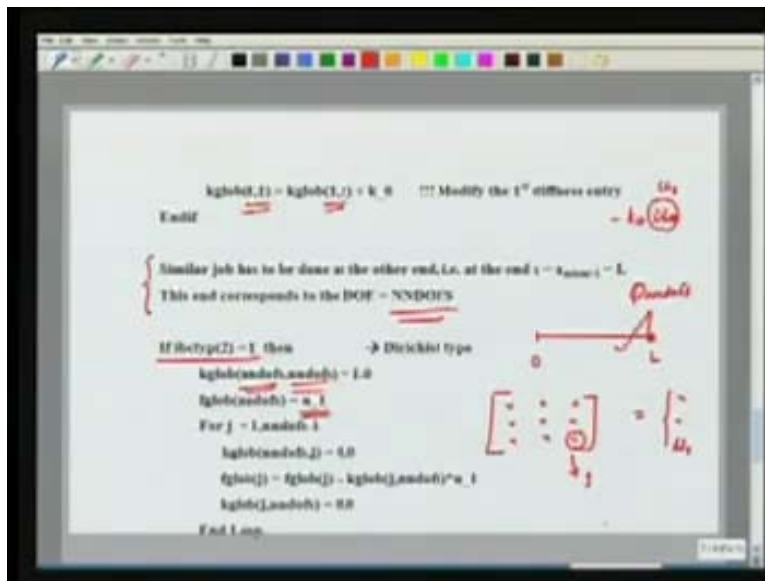
I loop over the rest of the columns in the first row that is, column 2 to column n_{ndofs} . I loop over these columns and I simply set the value of these column entries to 0. That is, I am going to set this one to one and everything else I am going to set to 0. So my first equation gives, this is equal to u_0 . So I get u_1 is equal to u_0 automatically. Since in the remaining equation also u_0 is setting there, I set this 0. I would like to now keep my stiffness matrix symmetric. I could have left it as such but by leaving these stiffness entries which are not equal to 0, I am destroying the symmetry of the stiffness matrix. To make the stiffness matrix symmetric, I simply eliminate these known degrees of freedom from the remaining equations. So I go to the remaining rows, the next row I know the value of the u_1 is u_0 . I simply remove this part from here.

By first changing $f_{j,1}$ corresponding to that entry that is from second, third, fourth up to n_{ndofs} , to what it was originally, minus this part due to the stiffness term multiplied by u_0 . This is known. So I subtract that and now this is the modified load entry from 2 to n_{ndofs} then I go and set the entries in the first column for all the rows remaining rows to 0. I set it to 0; that is, I have removed the influence of u_1 and the influence has been carried over to the load vector side by writing f_j is equal to $f_j - k_{j,1} u_0$. This is how I am going to take care of the Dirichlet boundary condition applied at point 1 and this also retains the symmetry of the matrix that we have.

Next is if I want to have a Neumann boundary condition at this end. That is at this end (Refer Slide Time: 31:54), I have a force Q applied; Q is tensile. In that case, life is very simple, I simply go to the first load vector entry and to which from which I subtract Q . This modifies the global load vector entries, so the first load vector entries and I have taken care of Q . This is what we needed from our variational formulation.

If I want to have a mixed type of a boundary condition that is the Robin; that is the end has a spring with spring constant k_0 and an initial compression of δu_0 that is, it is initially compressed by δu_0 . If it is compressed then this part is going to apply a force of $k_0 \delta u_0$ in the other direction, minus $k_0 u_0$, so we will take care first of this part. So the total force due to this end will be in this direction $k_0 \delta u_0$ minus u_0 . This part is a known and this part is an unknown. The known part we are going to add to the load vector side that is it becomes f_{globel} plus $k_0 \delta u_0$ because this is the compressive force. Now we have to add the corresponding unknown u_0 part to the stiffness side.

(Refer Slide Time: 33:44)

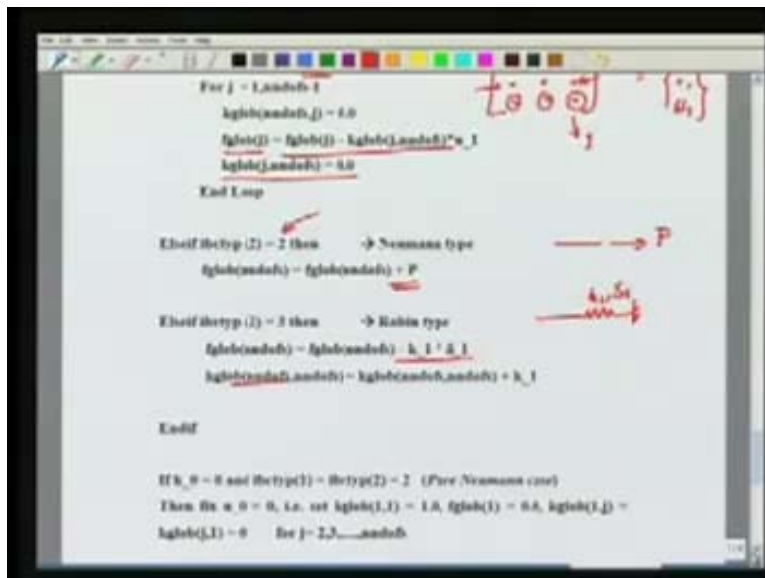


Since we had the part $-k_0 u_0$, u_0 is nothing but u_1 , I take it to the left hand side and I add k_0 to the first diagonal term of the stiffness matrix that takes care of the part due to this spring. I modified the stiffness part as well as the load part in the case of a Robin boundary condition. I essentially end the application of the boundary conditions for point 1.

We have to do a similar job at the point x is equal to x any element plus 1 that is the point $x L$ at the other end of the domain, so this is L , this is 0. What does this is end correspond to in terms of the degrees of the freedom? This corresponds to $nndofs$ that is, the degree of freedom or the basis function corresponding to this point is 5 $nndofs$. If I have to apply the boundary conditions at this end, I have to essentially modify the rows and columns corresponding to only $nndofs$. Again at

the end two, if the boundary is a Dirichlet type then, I specify what is the known displacement is that has to be applied here that is given by the value 1. Again I go and set the diagonal entry here like this (Refer Slide Time: 35:35). This diagonal entry which corresponds to nndofs to a value 1 and I will set the last load vector entry to the value u_1 , the specified value at this end. Again the rest of the entries in the last row, I am going to set to 0. So the last row is going to give me the value of the end deflection is equal to this specified value and again in order to obtain symmetry, these entries have to be suitably modified. The global vector for all other rows that is all the rows above will be modified by these entries, last row and last column. $fglobe\ j$ is equal to $fglobe\ j$ minus the stiffness entry corresponding to the last column. In that stiffness entry, I am going to take that stiffness entry multiplied by the known displacement value at this end and simply subtract it from the load vector corresponding to that row and then I will set the corresponding stiffness entry to 0.

(Refer Slide Time: 37:18)

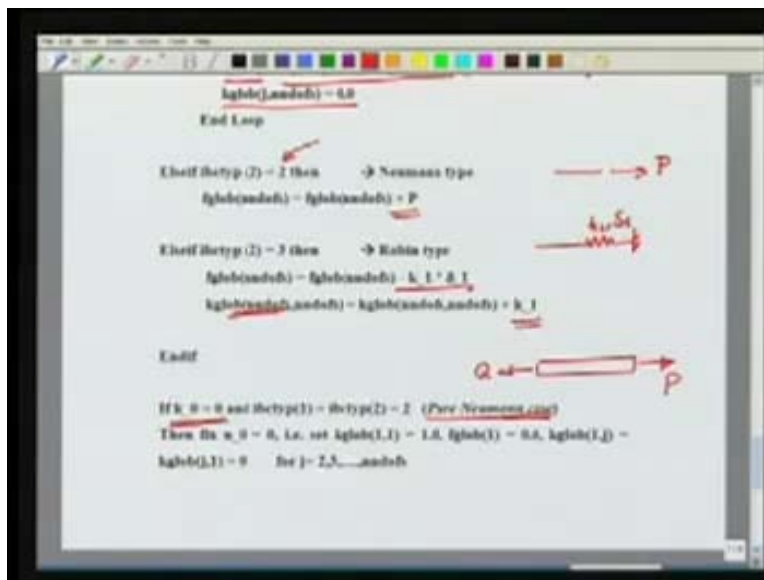


Similarly, if I have to apply the Neumann boundary condition here. For the Neumann boundary condition at this end, there is an end load P acting here (Refer Slide Time: 37:20) then that load has to be directly added to the last entry in the load vector. If I have a mixed condition here; here I have a spring, I call it stiffness k_1 and an initial compression δ_1 . The global load vector gets modified by $-k_1$ into δ_1 and the global stiffness matrix corresponding to this last column and

the last row that is, the last diagonal entry gets modified by k_1 . This is all we have to do as far as applying this boundary condition is concerned.

One more thing we have to check. In this case for the Robin boundary condition, we have the end spring k_1 and the end compression δ_1 . This again can be applied similar to what we have done at the end x is equal to 0 by first taking the last load vector entry from which we subtract this k_1 into δ_1 is the part of the load and then to the last diagonal term in the stiffness matrix we simply add k_1 . In this case, always the stiffness entries are going to get an addition of a spring constant. We cannot be subtracting, if I am subtracting something is wrong there.

(Refer Slide Time: 39:00)



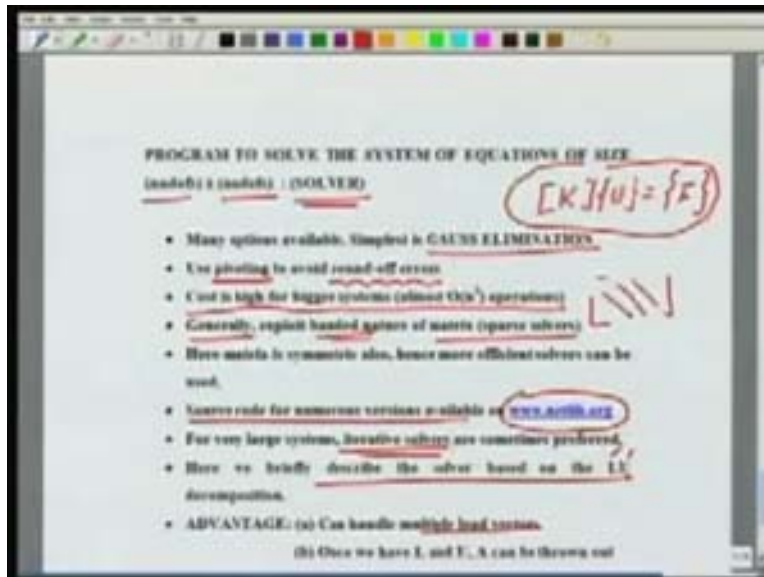
One more thing that we have to keep in mind is the special case, when there is no distributed spring from the structure. That is I may have this kind of a situation. There is no distributed spring and both ends have Neumann boundary conditions with or without point loads. That is not a problem. If the spring constant is 0 which I can check very easily and if the boundary condition type at both ends is 2, that is the Neumann case. So this is called a pure Neumann problem. In this case, if we think of this problem as I have this pen (Refer Slide Time: 39:45). Here is the load; here we are talking of static equilibrium. This will remain in static equilibrium but it can have a rigid body mode. This is going to reflect in our calculations; we will see that the stiffness matrix is going to be ranked deficient by 1.

There is a rank deficiency in this stiffness matrix of 1 and this corresponds to the rigid body mode. I take any solution to which if I add a constant, I get some deformation or displacement u for the various points. To this displacement, if I add a constant uniformly then that also is the solution because the state of strain and state of stress is not going to change, if I add a constant to the displacement field and this constant has not been constrained at **rest**. Numerically, we are going to see the effect of it through this rank deficiency and we have to fix this constraint. That is, we have to fix this rigid body mode to a particular value. Generally, we are interested in the strains and the stresses and may be relative displacement of 2 points. So why can not we fix this constant to an arbitrary value? We fix essentially may be this end point (Refer Slide Time: 41:15) to a value 0 so u at 0, fix it to 0; we find a solution. We see that this solution is not going to give any different state of stresses strain. This is because loads have to be in static equilibrium.

Remember one condition that has to be checked for the solution to exist that given this load P , Q and distributed F , these P plus integral of F over the length minus Q has to be equal to 0. That is there is global equilibrium for the whole structure. If that is not true, we cannot have static equilibrium for this particular case. This thing (Refer Slide Time: 42:00), P minus Q plus integral of $f dx$ from 0 to L is equal to 0 is called the consistency condition; the load has to satisfy. If that is true, then if I fix arbitrarily the value of the first point to 0. I do it as if I am applying Dirichlet condition at the point 1. That is, again I am going to set the first diagonal entry of the stiffness matrix Q_1 . The first load vector entry to 0 and all other entries in the first row and the first column to 0. I solve the problem; I get a solution which is going to be fine.

Remember this has to be done. If I have a distributed spring; say somehow it is lying on an elastic support then automatically, the rigid body mode is constrained against because we cannot have a constant deflection for the bar in this case. If we have a constant deflection then the spring is going to have a force and that is going to counter everything. It is going to counter the effect of the other force, if we have the constant. In this case, we have the constant but the stiffness matrix is not going to be singular; that one should check that, in case when k_0 is not equal to 0. I have the same problem as I have done here the stiffness matrix should not be singular. This is a check for the program. We have now applied the boundary conditions also. Next part is we have to solve the problem.

(Refer Slide Time: 43:58)



The matrix problem $K U$ is equal to F . There are many options which are available to us. One of the options and size of the problem is $n \times n$. This feature of solving the problem will be available through the routine called the solver; I could call that anything. There are many options available to solve this kind of a matrix system. Remember this system has very special properties. It is symmetric and now it is positive definite, so it should be invertible. One simplest possible way is the Gauss elimination in which one can see in any of the numerical analysis book. In the Gauss elimination, remember again from the solving point of view, we have to be careful. We have to use pivoting. One should go and read up what do we mean by pivoting. Why do we need by pivoting? Pivoting is to avoid round of errors. Here we are talking of large systems with varying sizes of the entries in the matrix, round of errors can happen. The problem with this Gauss elimination, though it is very nice and robust, if I use fully pivoted one is that the cost of computation is high. In fact most of the direct solvers which is by simply doing an elimination, do have very high cost of solution and it is of the order of n^3 ; n is the number of the size of the system; so it is n^3 .

If n is increasing the size is becoming bigger and bigger, the computational cost is going to become bigger and bigger, but here (Refer Slide Time: 45:50) we have a banded nature. That is, only in the vicinity of the diagonal, the entries are going to be non zero; elsewhere they are going to be 0. We can use that and we essentially get n^2 kind of operations. These are so called

parse solvers. Many such solvers are available freely; the source code of these solvers in Fortran, C, C + plus and so on.

One very good source for this is the website www.netlib.org. For very large systems we use something called iterative solver that is, here we do not eliminate but we make a guess and then try to improve on this guess solution. The advantage is there is that we do not have to invert the matrices. We have to do simply do matrix multiplication we are not going to talk about these iterative solvers but these are pretty big an important emerging field. Especially because we want to consider more and more complex problems which need significantly large number of degrees of freedom in order to obtain a solution.

In the next lecture, we are going to talk about so-called LU decomposition based solver. Sometimes it is called Cholesky decomposition and the advantage with this, it is very much similar to gauss elimination but it is simply managing information in a subtle way such that we can easily handle multiple load vectors. Once I have this kind of a decomposition of A (Refer Slide Time: 47:30) then we can **throw a variant** and all future computation can be done with LU.

I am going to stop here. In the next lecture, we will talk about the solver. Once the solver is there then essentially our processor has come to an end. Once we have obtained the solver, we are capable to get a solution. Once we have the solution too, now we do what with it? Whatever comes under this 'doing what' with the solution is in the module of the post processor. Here, we will simply see how to visualize the solution. What is it that we have to do with it and how to get better stress values out of finite element data, etc.