Finite Element Method

Prof. C.S. Upadhyay

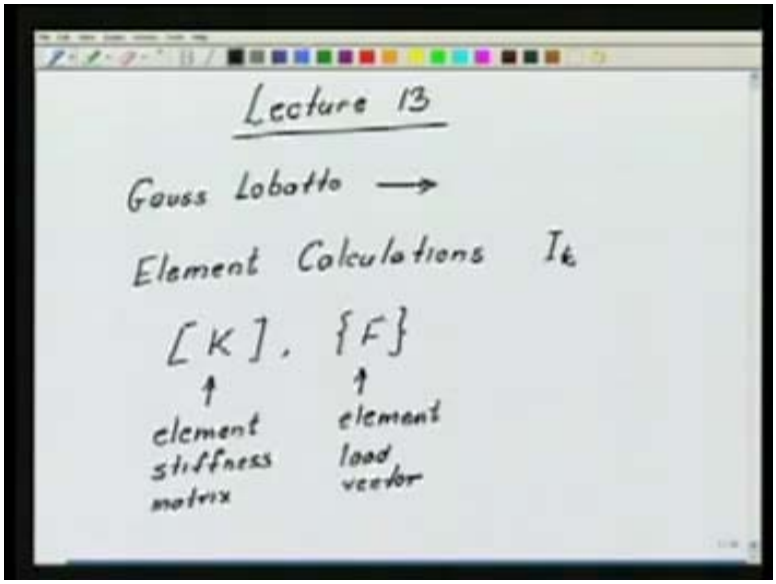
Department of Mechanical Engineering

Indian Institute of Technology, Kanpur

Module – 4 Lecture - 3

In the previous lecture we had looked at how to do the numerical integration for an element. We have outlined a generic way of doing the gauss Legendre integration; one could choose a different integration rule.

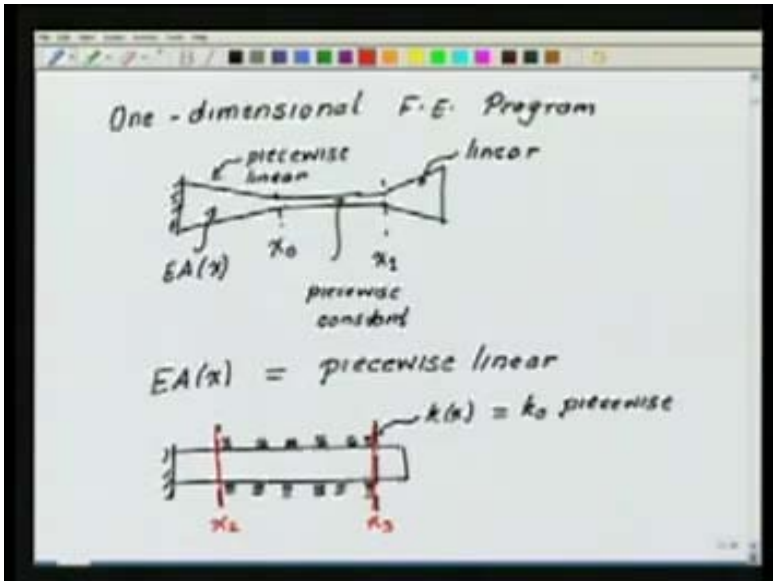
(Refer Slide Time: 00:22)



For example, there is another integration rule available which is called Gauss Lobatto but that is generally not used. This has its own features. If we can fix the integration rule for an element based upon the order of the integrant involved then we can do the element calculations so here we are going to concentrate on how to do the element calculations. For a generic element I_k, we will have the stiffness matrix K and the load vector F, for the element. We would like to get the entries of this element stiffness matrix and the element load vector. Before we go and do the

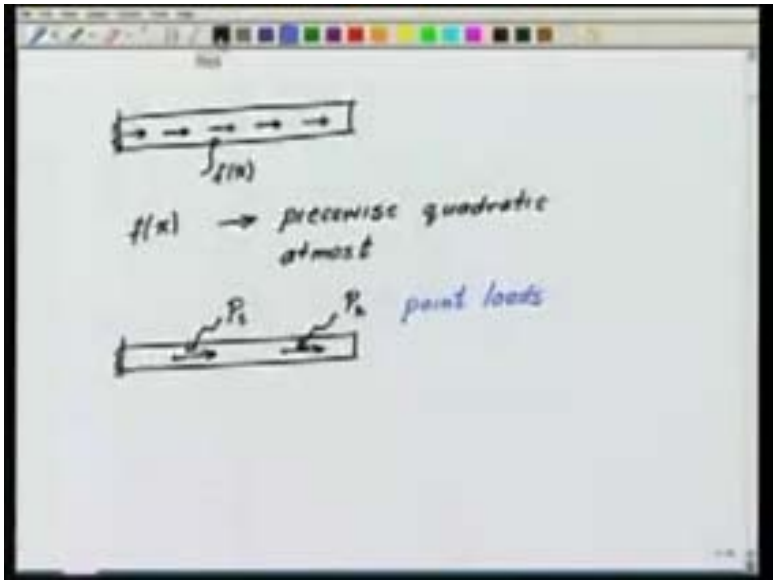
process by which we will obtain the entries of the element stiffness matrix and the load vector, let us fix certain ideas; fix essentially, what our finite element program will do.

(Refer Slide Time: 02:36)



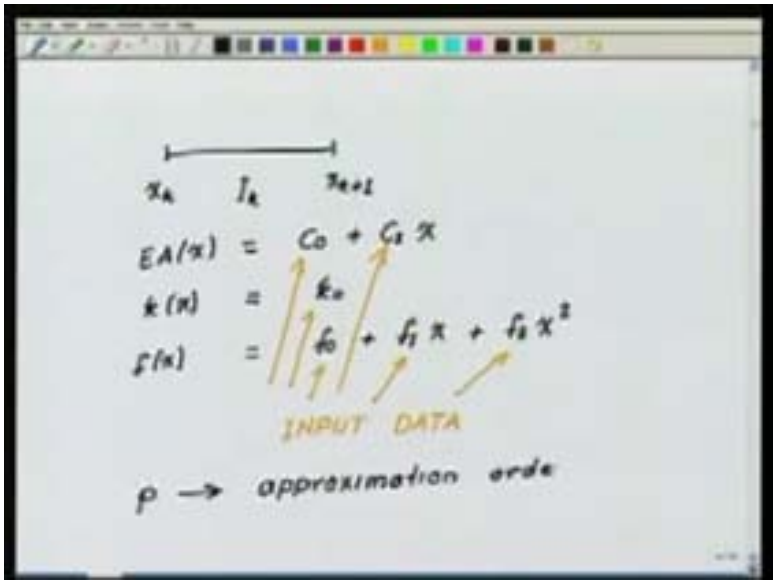
From the fine element program, we want the following things to be possible to handle: we may have domains with different material coefficients or material constant in different parts of the domain. Let us say this (Refer Slide Time: 03:20) is a point x_0, this is a point x_1 in this the material constant is piecewise linear, here EA is piecewise linear here EA is constant, here also it is linear; so as far as our EA is concerned, will say that this is at most piecewise linear. Second thing we would like to do is, given a bar we would like this bar to also lie on elastic support, which is a continuous elastic support. It has constant elastic constant k(x) is a constant piecewise. We can imagine that in a particular part of the bar, I have the elastic support; in the rest of it, it is not. So again this will be demarcated by some points, let us say, x_2, x_3.

(Refer Slide Time: 05:37)



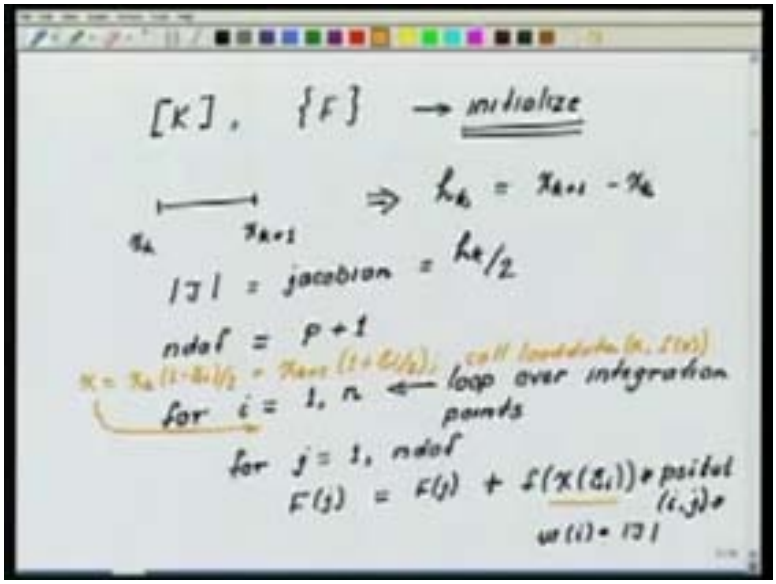
As far as the distributed body force is concerned, again lets have the bar (Refer Slide Time: 05:40) over which I am applying this distributed body force f(x), this can be f(x) is piecewise quadratic at the most. If we are really interested in making our code little bit more general then we can also do the following to this bar; it need not be uniform cross section material. I will also apply these point loads in the interior; P_1, P_2 these are point loads.

(Refer Slide Time: 07:09)



This is what we want our code to be able to handle. With this in mind, in an element if I take a generic element I_k with end points x_k and $x_{k \text{ plus } 1}$ I know that in the element EA(x) is some constant plus some linear at the most. The spring stiffness is a constant in this element and f(x) is at most a quadratic given by this data. These values of the coefficients f_0, f_1, f_2, K_0, C_0 and C_1 these are going to be input, they should be available to us as known data. This f_0, f_1, f_2, K_0, C_0 and C_1 has to be obtained piecewise which will come from somewhere. Using this information now we can go ahead and do the element calculations. Let the order of the approximation which we say is going to be fixed for the full domain is given as P.

(Refer Slide Time: 09:24)



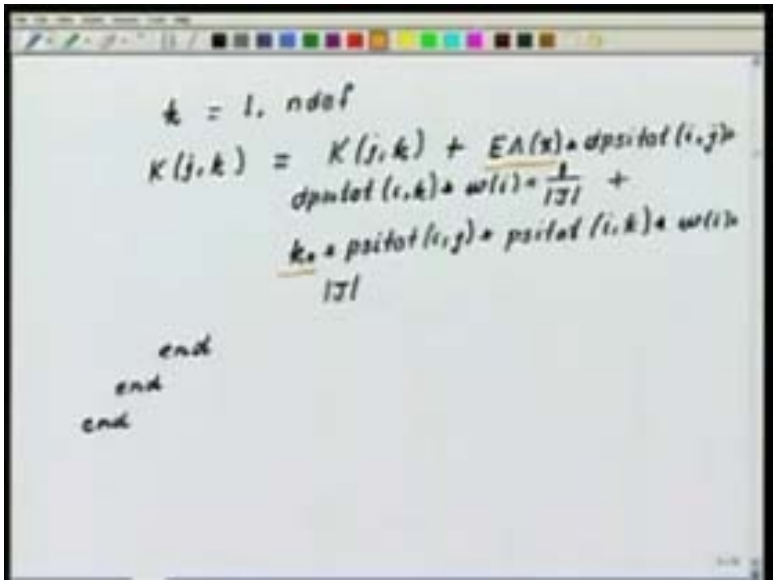
As far as the element calculation is concerned, I am going to take this element matrix K and the element load vector F and I am going to initialize it. That is, I am going to set the values of all the entries of K and F to 0 before I start doing anything. Since, x_k, $x_{k \text{ plus } 1}$ the two coordinates, two end points of the element are known they will be available to us. We can find the size of the element is equal to $x_{k \text{ plus } 1}$ minus x_k given the size of the element, we can find the size of the Jacobian is equal to h_k by 2. I am following the steps in the way we have to do things in a computer program. Then given P, I will define the number of degrees of freedom in the element is equal to P plus 1 and I give it a name ndofs. Given the size of the element, given the j, now I start loop over the integration points (Refer Slide Time: 11: 34).

Once I start the loop over the integration points then for each integration point I start the loop over the degrees of freedom of the element. In this loop I am going to do the following, I am going to say F(j) is equal to F(j) plus, now I am going to do the numerical integration to find my load vector entries, so I take the j th load vector to this I am going to add the contribution due to this particular integration points. It is going to be f(x) at the point psi_i into psitot.

For the ith point in the jth location, into these two things are there, then multiply with w corresponding to the ith point into the Jacobian. Here if we see this x of psi_i how will I get this? f

x of psi_i so this part for this I am going to now add this extra bits here now for the integration point, I am going to compute x is equal to x_k into 1 minus psi_i divided by 2 plus $x_{k\ plus\ 1}$ into 1 plus psi_i. This gives me the physical coordinate corresponding to this integration point. Given this x I also need to obtain f(x) so I will obtain it from some routine called load data to which I will pass the x and it will be turn me f(x), so this is the routine; for each integration point I find physical coordinate corresponding to this master coordinate. At that physical coordinate I take that value pass it on to this load data routine which should for this element return the value of the distributed load at this point. That I put it in here (Refer Slide Time: 14:52) multiplied by the value of the shape function corresponding to this point into the weight function into the Jacobian. So this is all I have to do as far as the load vector calculation is concerned.

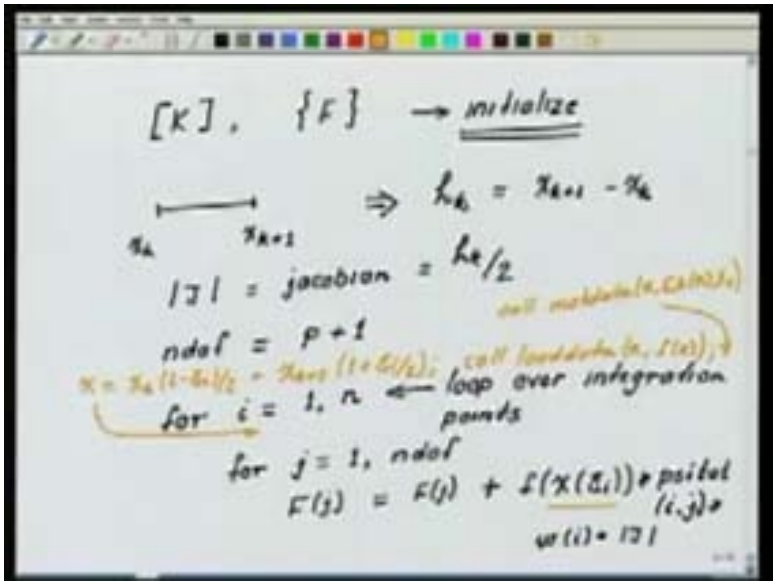
(Refer Slide Time: 15:11)



For the stiffness calculation, again I have this loop over the degrees of freedom and here (Refer Slide Time: 15:30) I am going to say that K (j, k) so the entries corresponding to the jth row and the kth column of the stiffness matrix for the element is given as K (j, k) that is the previous value that I obtained plus the part due to the EA. So EA at this point x into the derivatives of the shape functions; I will have psitot for ith point into the weight corresponding to this point. Remember that this was multiplied by 2 by h_k. That will be 1 by Jacobian; this (Refer Slide Time: 16:50) is the part due to the material as such. The part due to the distributed spring which

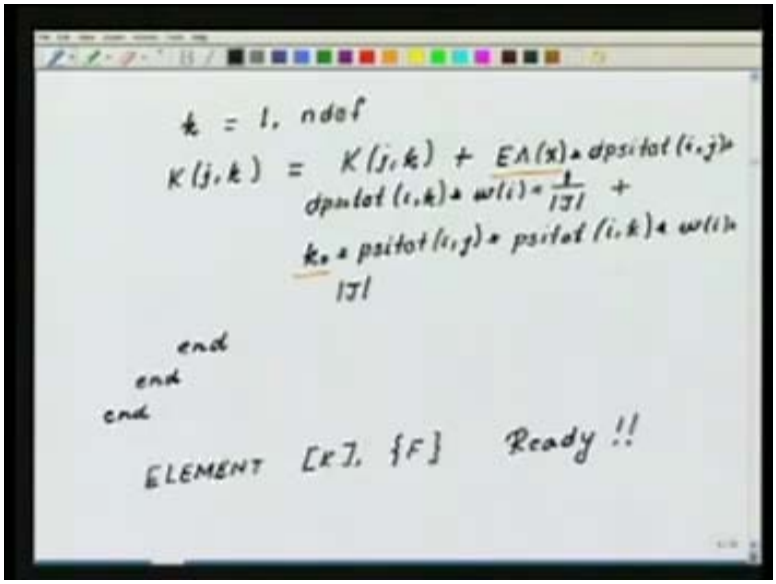
is attached if it is there in the element; it will come out to be K_0 into w (i) and multiplied by the Jacobian. We do the end loop over the K, end loop over the J, end loop over I. If I do this I have obtained the entries of the stiffness matrix and again the question is I need this and I need this (Refer Slide Time: 17:47).

(Refer Slide Time: 17:46)



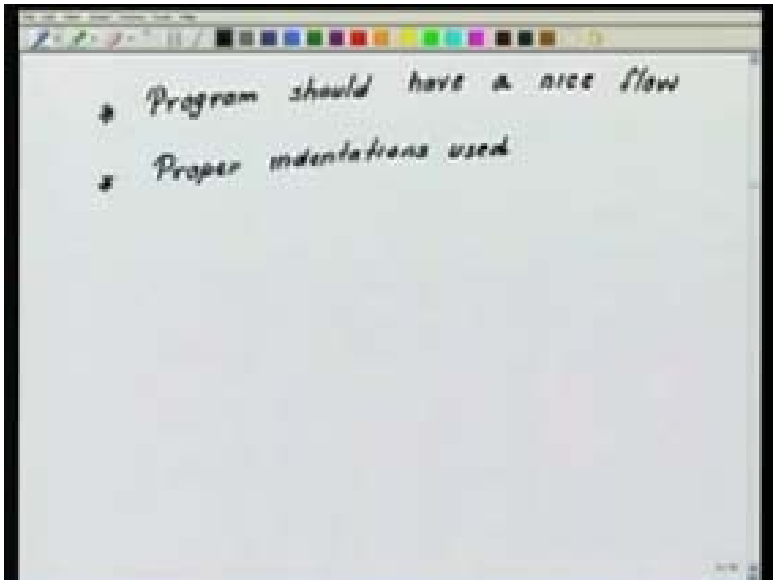
For this what should I do? I should go back up there and here after I compute x, I call load data, here I also call material data which will take the x returning the value of EA(x) and K_0. I will have to call the load data for the particular x both the load sub programs which will return the value of the force f(x) at this point and also the material data sub program which will return the value of the EA and the K at this material point.

(Refer Slide Time: 09:03)



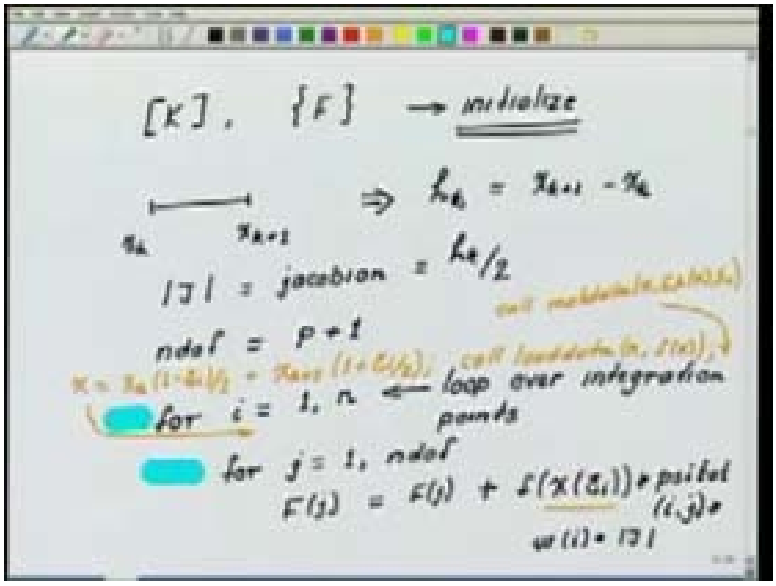
I do the integration and element stiffness and load vector is F this is ready. This is all we need to do as far as the element calculations are concerned.

(Refer Slide Time: 19:29)



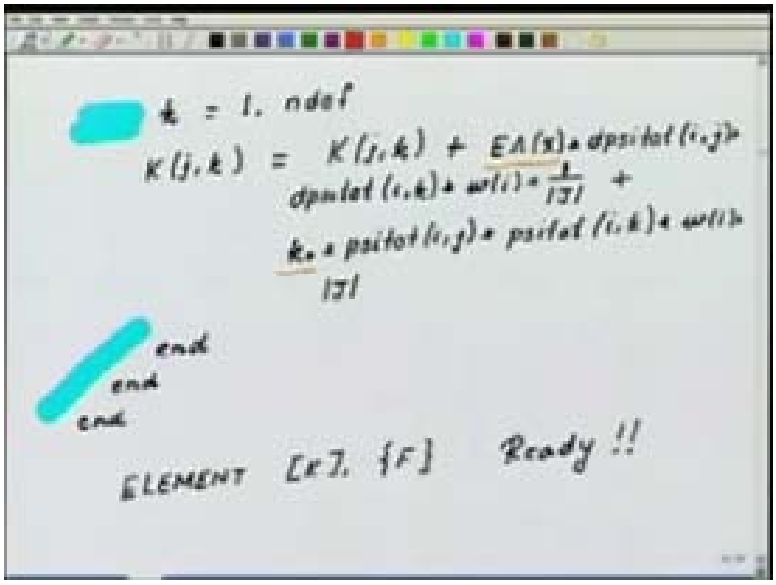
One thing that we should do, in order to avoid making errors while programming, is that the program should have a nice flow. Secondly, in a program proper indentations should be used. What do we mean by proper indentations? If I go back and I look at these loops that we have introduced one loop over the integration points, the other loop over these degrees of freedom the next loop over K.

(Refer Slide Time: 20:19)



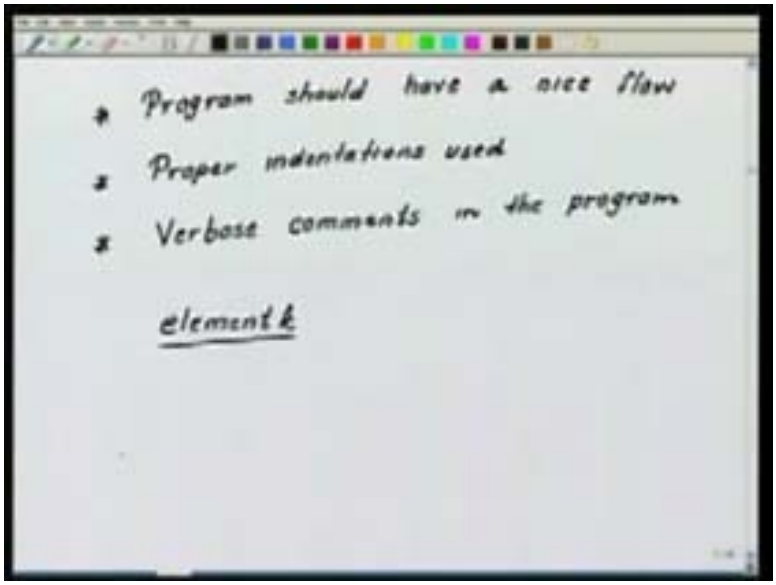
We should have the embedding of loops coming out clearly through the indentation of the statements.

(Refer Slide Time: 20:29)



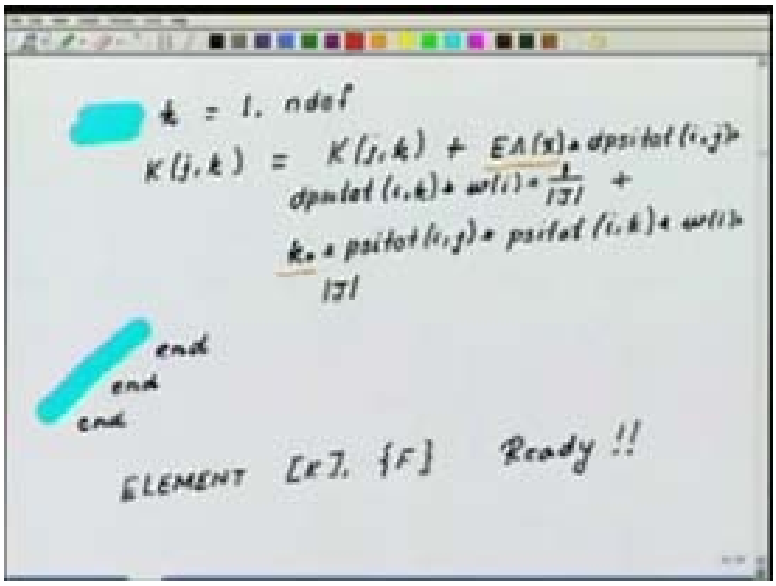
If this loop was innermost loop it is indented the most. If we have this kind of structure and in the end we see that this indentation also comes out as far as the ending of the loops is concerned it becomes very clear that we have, which loop is embedded inside which one and where the loops have ended because otherwise it can lead to problems.

(Refer Slide Time: 21:05)



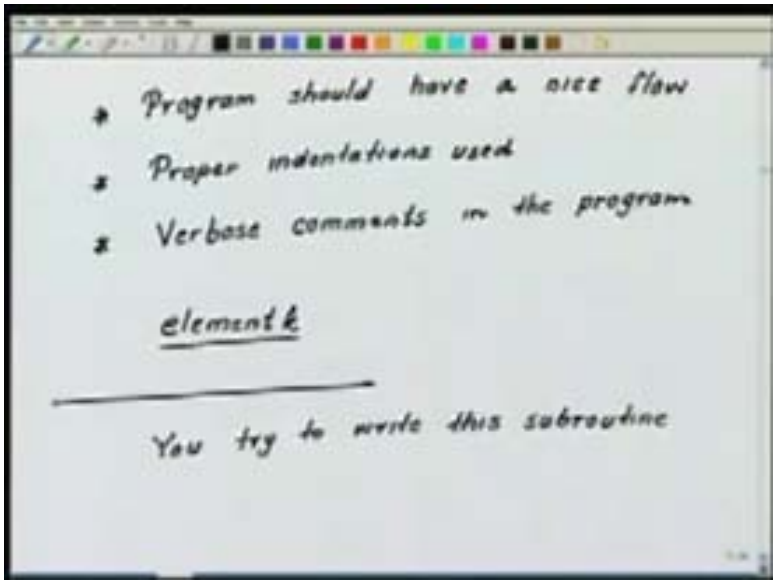
The other thing we have to do in all these programs for clarity and ease of reading and ease of use, has verbose - very verbose comments in the program. Once all these things are taken care of most likely the program that will come out, the routine or the sub routine will be error free. I will name these element calculation routine as Element k; this is the name of the routine. We see that I would like to give names which are relevant; I cannot call the sub routine which does element calculations something like epsilon, delta and so on.

(Refer Slide Time: 22:19)



Similarly, when I have these variables, the variable names should not be arbitrary they should reflect what they really correspond to. If I am talking about stiffness I should write it as K or K stiff. I can use more words if I am talking about load vector I can write it has x load or F load something like that; shape functions psi, EA material constant K, we can say K springs Kx and so on all those things. The naming should be properly done.

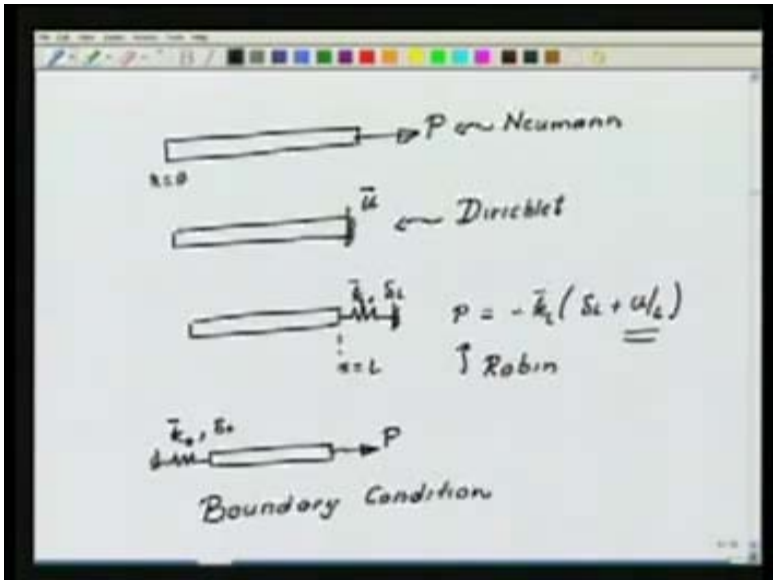
(Refer Slide Time: 22:51)



Our element calculations are over; as an exercise, we should try to write this program, write this sub routine. As far as the full pledged program is concerned we would like to write what are the other things that we need? We have specified that as far as the material is concerned we would like to have piecewise linear material, EA as piecewise linear K as piecewise constant, the load is piecewise quadratic and I can have point loads.

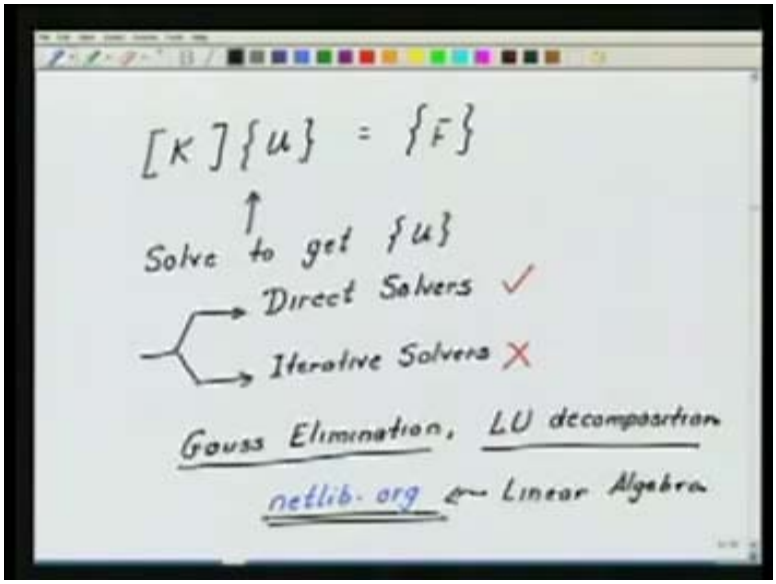
Similarly, P we said the order of approximation is going to be uniform throughout the mesh throughout the domain. We would also like to fix, what kind of boundary conditions we would like to impose. Till now, I am not being very general in what I did. We would like to have certain types of boundary conditions.

(Refer Slide Time: 24:10)



Let us look at the kinetic boundary conditions; at an end let us say a particular end (Refer Slide Time: 24:20), I can give a force, I can give a displacement, it can be built in or some specified value so u is specified or I can give we the force in terms of the displacement, this is a spring with constant k bar and an initial compression delta L. If I look at this spring, this is at end x is equal to L (Refer Slide Time: 25:00). Here we see that P equal to minus k bar into delta L plus u at L. So here the load, the ends load actually in terms of the end deflections. All these cases I would like to have, I would like to have this one this is Neumann condition, this is a Direchlet condition and this is called Robin condition. These things we would like to have in our code and according to the user's desire I should be able to apply at each of these ends. What I can do at this end x equal to L can be done at the end x equal to 0 also; whatever I choose I can do. For example, I can have this kind of a situation (Refer Slide Time: 26:05), here it is k bar at L let us say this is k bar at 0, we can have an initial compression delta 0 and this end, I have an end force P. So I can have any combination of the two things so that will decide what boundary condition I would like to apply, so the boundary conditions that we want are of this type.

(Refer slide time 27:00)

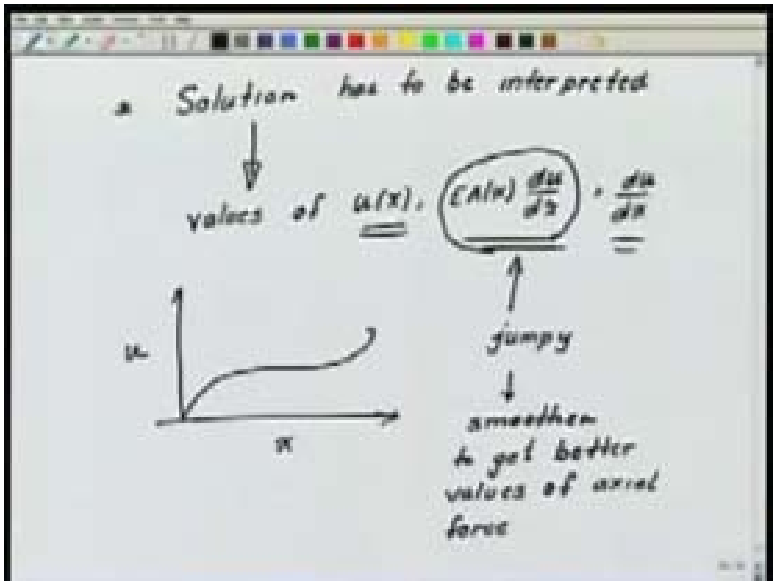


Another feature we would like to have in the code is something that we will discuss in the future. I have now obtained the global matrices K and the global vector F. I would like to solve it to obtain the unknown coefficient U. To solve, there are various ways of solutions and those can be classified into two broad classes or families, so-called Direct Solvers and Iterative Solvers. In the direct solver, I try to do essentially elimination to obtain the solution U directly. In the iterative solver, I start with a guess value of U and I try to minimize the difference between K U minus F, that is called the residue, till I converge to the solution. Here we are not going to look at the iterative solvers we are going to look at the direct solver and in direct solver we are going to look at simplest direct solvers; two types which are based on the Gauss Elimination and something called LU decomposition. We outline the algorithms for both of these processes.

One should also be aware that these are available freely on the net, so there is really no need to go and write these solvers by ourselves. There are some websites for example, a website called netlib dot org which has a huge repository of linear algebra routines. It has all sorts of programs available, sources codes are available. One can download these things for Fortran and C Fortran 95 and those versions from this website. This is a very useful website from which these solvers can be downloaded in fact better solvers can be downloaded which essentially take advantage of the sparcity of the stiffness matrix or the fact that the stiffness matrix is banded. Let us now

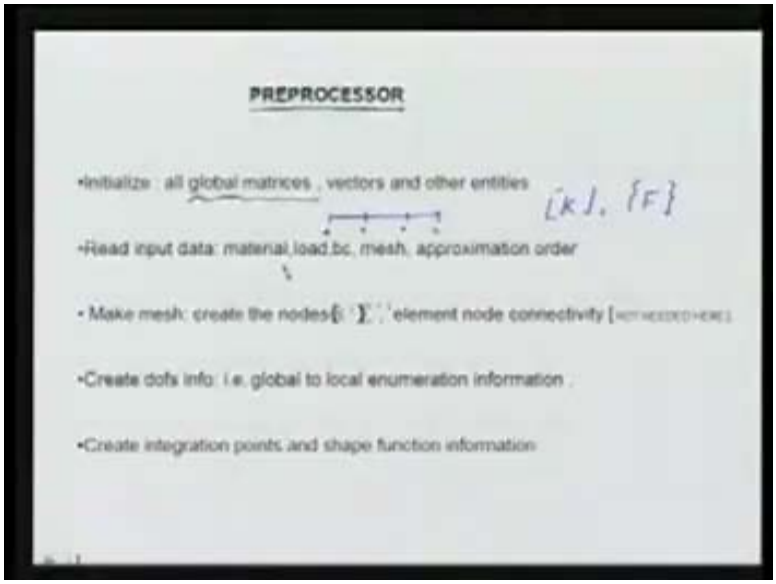
recap what we like our program to do. First thing we said that these are the input data that the input data which we wanted to specify, we have specified, what boundary conditions can be applied. I obtain a solution using this solver.

(Refer Slide Time: 30:33)



This solution has to be interpreted; how do we interpret this solution? We would like to know various things; the value of u(x), EA(x) du dx at points simply du dx at points; given a point I like this value, or I would like to plot out these things. For example, u is there, x is here, so I would like to have a plot this (Refer Slide Time: 31:20). We would like to use our vector u to construct the solutions u or these quantities wherever I wish. This comes under the category of post-processing of the information that is coming out of the finite element computation. There are other things that the finite element data as such, the finite element solutions will give us these forces which are not continuous, which are jumping in nature. We would like to smoothen these out to get better values of the resultants of axial force. With this I will like to outline the basic structure of the finite element code, the one dimensional finite element code that we are going to write. Our finite element program will have three basic modules, bigger modules. The first module of the element of the finite element programs will be called the preprocessor.

(Refer Slide Time: 32:50)



It will be called the preprocessor that have here. In the preprocessor what are the things that we are going to do? In the preprocessor first thing that we do is we go and initialize all the global matrices. We will initialize all the global matrices. What are the global matrices that we have? The global matrices that we have are the global stiffness matrix K and the global load vector F that has to be initialized. We have our vector which gives us the coordinates of the points in the mesh, or the nodes, these coordinates. The vector of the coordinates also has to be initialized. Our integration points vector has to be initialized; our weight functions vector has to be initialized and then these arrays which carry the values of the shape functions of the integration point and the derivatives of the shape functions of the integration points have to be initialized. It is always a good idea to initialize all the global matrices, arrays, and vectors right at the beginning of the program. Also if I am using C or other things the entire global variables initialized too.

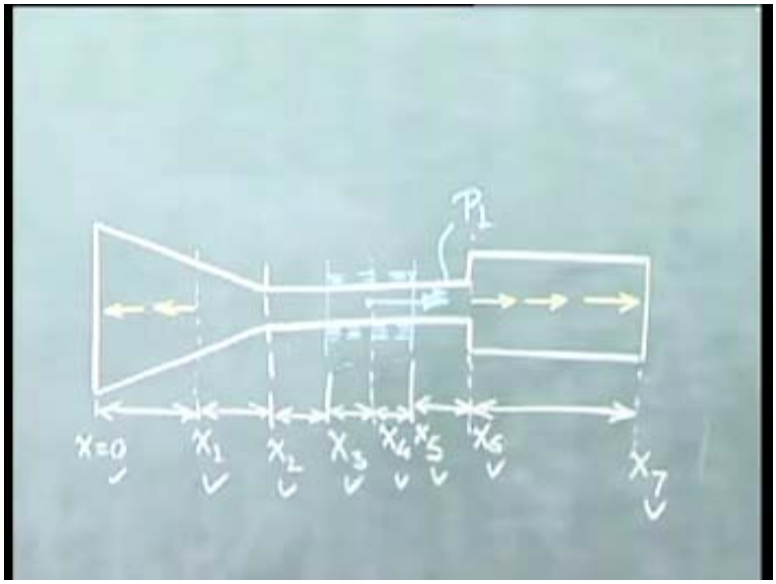
We have to read the input data, what are the input data that has to be read in? First is the material data. How many material domains is the mesh broken into? And for each of the material domain I may this many partitions of the domain and in each material domain give me the material properties. How are the material properties given in terms of the coefficients of the EA(x) that is EA(x) is equal to C_0 plus C_1 x in a particular material domain; so in each material domain these

values have to be given? Similarly, the K_0 has to be given each of the material domains. Similarly, I will have to give the load data that is the distributed load, how is it varying? Is it varying in the whole domain as linear, quadratic, and constant or is it applied piecewise in a small region? It is something in the other regions, is it something else? Everything has to be given upfront - boundary conditions.

At each of the two ends what kind of boundary conditions is applied? For example, (Refer Slide Time: 36:00) at this end x is equal to 0, I may have a Dirichlet type boundary condition or a Neumann type boundary condition or a Robin type boundary condition. For each of these boundary conditions I have to give the value of the corresponding parameter, which is needed. For example, if it is Dirichlet then I have to give you what is U at 0 as an input data; if it is Neumann, I have to give what is the tensile force at this end; if it is Robin, then I have to give the spring constant of the attached spring and the initial compression of the spring. This has to be done at both the ends and all that information has to be read in from an input file into a program. Given all these things also I may be having this point load data that is there are certain points where I would like to apply point loads. This list can be endless. We have to restrict ourselves to a specific class of problems that we would like to handle as a sample case.

Mesh for example; we may take the mesh, to be made region by region because the mesh has to honor the material domains, the boundaries of the material domains. I know that at a point where a point load is applied I have to have a node. These things this information has to be built into the code. So when I made the mesh for the mesh I have to give the information as to what are the boundaries of each of the sub domains over which the meshing has to be done and then in each sub domain I have to say how many sub divisions we want. Let us say in each sub domain we want uniform meshing. Let us say in the first sub domain I want five sub divisions; it will put five uniform elements here and in the next sub domain may want two, so we will tell two so it will put two elements. Accordingly it will construct, it will create these nodes which will be stored. In the third one I may have four elements, it will accordingly put the nodes which will divide this piece into four elements.

(Refer Slide Time: 38:23)

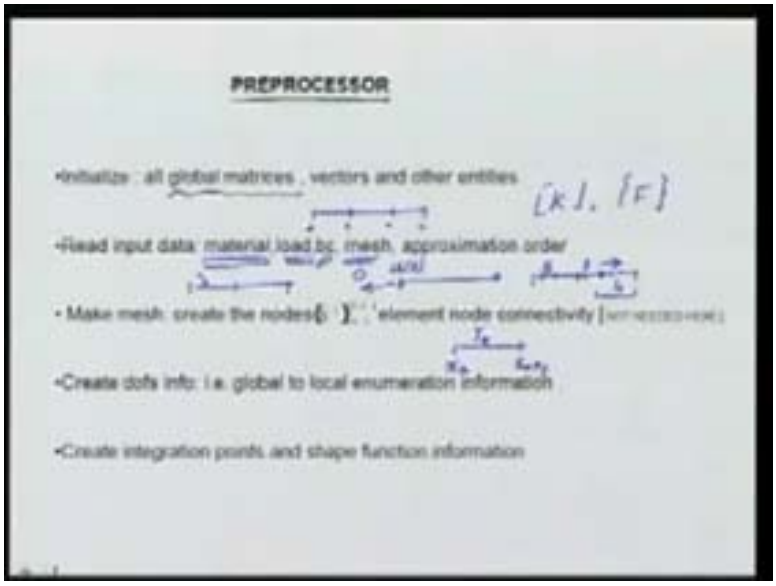


Let us take the example of a domain like this where I may have in this region, the distributed spring support, I may have a distributed force of some intensity in this region and I may have some other intensity in this region (Refer Slide Time: 39:25). Further, I may have points loads somewhere here P_1. So as far as the making the mesh is concerned I will have the first sub domain going from point x is equal to 0 to some point x_1. The second sub domain - this is the sub domain where my load distributed load is applied and here ends and the load changes. The next one will be this one which will go up to the point x_2 so here what happens the material is one type but here it is changing; that is the cross sectional area is changing; I have to honor that.

Third sub domain is this part (Refer Slide Time: 40:39) where we are in this constant cross section part but there are no distributed springs; this will become my third sub domain. The fourth one we see that here the distributed spring is here but there is a point load is there I have to honor the point load. The fourth one is this x_4 then the fifth one from here I continue I see that this is the end of the region where the distributed spring is applied. This become $x_{5;}$ this continues till this one (Refer Slide Time: 41:27) where again the area changes at x_6. By coincidence, I have taken the distributed load of a different type but coinciding with the whole sub domain here. For that case, all these interface points, all these transitions points due to various reasons due to material, load and etc and etc have to be clearly specified in the input data

and the meshing has to be done piecewise. This will take care of in future lecture but this has to be honored. If I do not do this our own computation may give us faulty results, results which are not very good. This information is also true for two and three dimension where we have to honor the material boundaries. As far as the mesh is concerned, we have to honor those sub domain boundaries which come in due to the load data etc.

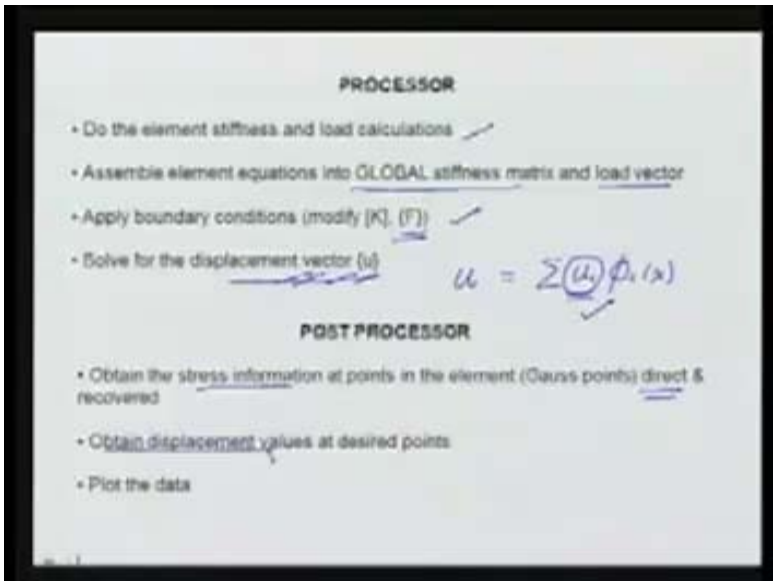
(Refer slide time 42:40)



Approximation order we are going to fix for the element, for the whole domain we will assume that the approximation order is the same. After doing this, after getting all these input data we will now make the mesh; that is, in each of the sub domains we have to put some number of elements as we desire. We will have to generate some information called the element load connectivity information; that is, which are the nodes which form the extremities of the element? That is very simple because for the element I_k I know the node given by x_k and $x_{k \text{ plus } 1.}$ So that is not a very big problem; but this is a feature which should be added in the finite element program because in two and three dimensions this becomes very important. The other thing that we have to do is, we have formed a mesh we have read all the input data, now given the approximation order we have to construct the degrees of freedom over the elements. We know that given P in an element there are P plus 1 degrees of freedom. We will have to do the construction over the whole domain that is we will have to construct the basic functions. To each element we will have

to assign which are the global basis functions which corresponds to the local degrees of freedom in the element this is called the global to local enumeration and this is the feature which would be there in any dimension we will work in. 1d, 2d or 3d. The next thing is, we have to create so we have constructed the mesh, we have constructed the approximation information. We have to construct the integration points and the shape function information for which we have already seen how to write the respective sub routines or sub programs.

(Refer Slide Time: 44:43)



This was the whole body of the preprocessor; essentially, the preprocessor gets us ready to go and actually start doing the actual computations with respect to the problem at hand. We go to the processor; processor is the heart of the program where the whole element calculations are done and the element stiffness and load vectors are created here, the element stiffness and the load vectors are created. These are passed to some place where these element calculations are assembled to get the global stiffness and load vectors like we have done in an earlier lecture. How do we assemble in a computer by using this local to global enumerations? This global to local enumeration array or a vector will tell us which global locations should our element entries go into.

We pick the right row and column in the global stiffness matrix and the right row global load vector where I will go and add these element contributions so this is the process of assembling. After global stiffness matrix and the load vectors are assembled, I will apply the boundary conditions at both the ends. What does it lead to? It leads to application of the boundary conditions it will either lead to a modification of stiffness matrix. If I have the Dirichlet boundary conditions then I have to enforce the value of the solution at the particular point is equal to the given value; that is done by doing suitable modifications to the stiffness matrix, as we have done in one of the assignments earlier.

Similarly, the load vector also has to be modified, if I have a Neumann conditions at end then I have to go and add the effect of that end load applied to the global load vector. If I have a Robin boundary condition then I have to modify both the stiffness and the load vector entries because from the Robin we see that the end loads comes in terms of unknown end displacement, so that unknown has to brought into the stiffness matrix contribution. For all these cases we have to appropriately modify the global stiffness matrix and the load vector to account for the applied boundary conditions. Once I take care of these things then I go and solve the final system K u is equal to F. Find out, what are the displacement vectors u.

The displacement vector u is essentially corresponds to the coefficients of our series representation of the approximate solution. If we remember that u approximate was u_i phi_i. This we have to always remember that this is going to be true irrespective of what we do. Once I have this representation, if have this coefficients, I have essentially the whole approximate solution in the whole domain wherever I need the solution I can go plug the value of x I know the coefficients I will get the u - is as simple as that.

In the element I know only the element shape functions or the active phi(s) all the other phi(s) are 0. We will take those element shape functions multiplied by suitable coefficients u_i and we get the u in the element; take derivatives and we will get the derivative of u in the element. From this we can construct all the information we need. In order to display the response characteristics of a components loaded by certain loads. This part of displaying and using this displacement vector that we have computed to get some meaningful response quantities goes into the third big module which is called the post-processor.

The post-processor is the most colorful part of the whole program but it does not too much the bulk has been already done. In post-processor we try to get what we would like to get obtain stress information at points in the elements; good points are the so called Gauss points which are the integration points. We can give the direct value of the derivatives and the stresses of the axial forces or we try to do some kind of smoothening or recovery to which we claim to get better values of the stress information. We can obtain displacement values at desired location like maximum displacement, displacement at a joint where two dissimilar materials are present. Interface of those we can plot the data we can use various plotting algorithms which are all available freely to plot the graph of the response we can have the graph of the displacement, graph of the slope, graph of the axial force all these information, we can have plotted in the post-processor. We can create various kinds of colour pictures which essentially forms the body of the post-processing part of the code.

Once we have these three modules ready, we have a finite element program. In the next lecture, we look at the remaining pieces of the modules. Remember that, the integration the shape functions and the element, we have already done. How do we fill in the rest of the part? That is making the mesh then creating the degrees of freedom information and this global to local enumeration, then applying the boundary conditions and so on. Solver, we talk of those, we talk of a simple post-processing scheme to get better stress information and then the program is essentially ready.

Thank you.