

Indian Institute of Science

Variational Methods in Mechanics and Design

Prof. G. K. Ananthasuresh

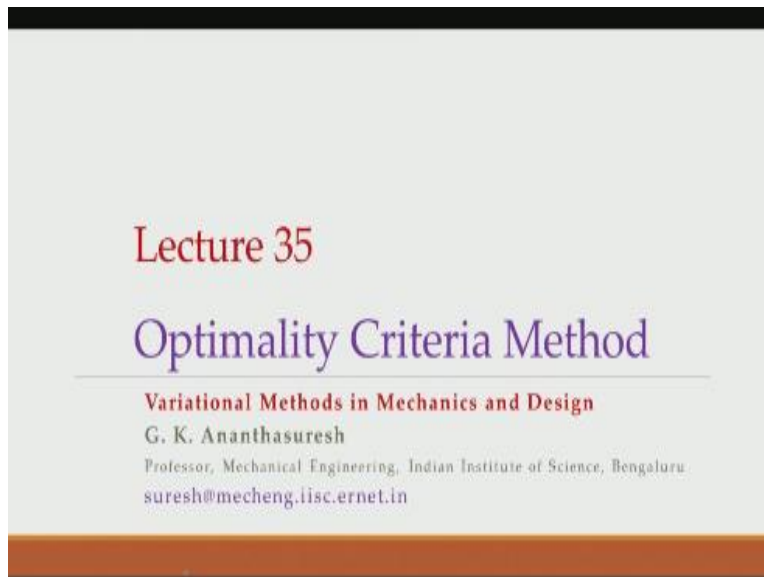
Department of Mechanical Engineering

Indian Institute of Science, Bangalore

NPTEL Online Certification Course

Hello we are going to focus on the last word of this class that is variational methods in mechanics and design we are going to discuss implementation of the algorithms that we have already discussed at length that is the optimality criteria method we will see it in action today using the software mat lab first we will go through the algorithm one more time and look at the code every line that is important for us to look at and then run the optimizations for a bar as well as a beam the stiffest bar and stiffest beam were given amount of material okay.

(Refer Slide Time: 00:59)



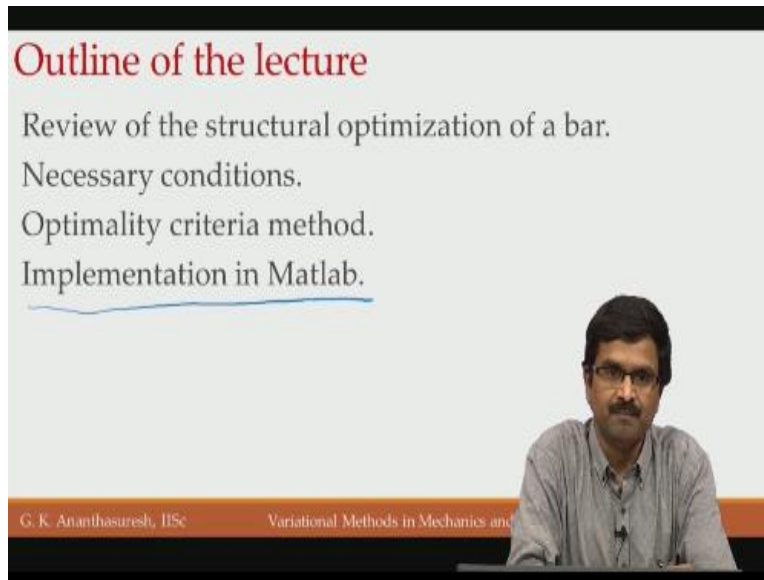
Lecture 35

Optimality Criteria Method

Variational Methods in Mechanics and Design
G. K. Ananthasuresh
Professor, Mechanical Engineering, Indian Institute of Science, Bengaluru.
suresh@mecheng.iisc.ernet.in

So let us look at the optimality criteria method that we are going to use.

(Refer Slide Time: 1:06)



We will first review the structural optimization of a bar problem talk about the necessary conditions and then describe the optimal getting a method in brief because we already discussed that and focus on the implementation which is our main task today.

(Refer Slide Time: 01:27)

Stiffest bar for given amount of material

$$\text{Min}_{A(x)} MC = \int_0^L pu \, dx \quad MC = \text{mean compliance}$$

Subject to

$$\lambda(x): \quad (EAu') + p = 0 \quad \checkmark$$

$$\Lambda: \quad \left. \int_0^L A \, dx - V^* \leq 0 \right\}$$

Data: $L, p(x), E, V^*$

So this is the statement of the problem where we are trying to minimize mean compliance if you recall MC is mean compliance it is basically work done by the external force and if you minimize it that is equivalent to maximizing the stiffness of the bar and this is the governing equation for the state variable u is our state variable $A(x)$ is our design variable on that design variable we have a resource constraint that is a volume constraint.

Where V^* is there so the data for this problem which we must write we know the length of the bar we know the loading $P(x)$ and we know the E modulus of the material and V know we start these are the things that we know okay.

(Refer Slide Time: 02:25)

Stiffest bar with a given volume of material

Min $MC = \int_0^L pu \, dx$

Subject to

$\lambda(x): (EAu') + p = 0$

$\Lambda: \int_0^L A \, dx - V^* \leq 0$

$$L = \int_0^L pu \, dx + \int_0^L \lambda \{ (EAu') + p \} \, dx + \Lambda \left(\int_0^L A \, dx - V^* \right)$$

$$\delta_A L = 0 \Rightarrow \Lambda + \lambda Eu'' - (\lambda Eu')' = 0 \Rightarrow \Lambda - E\lambda' u' = 0$$

$$\delta_\lambda L = 0 \Rightarrow (EA\lambda') + p = 0$$

$$(EAu') + p = 0$$

$$\Lambda \left(\int_0^L A \, dx - V^* \right) = 0, \Lambda \geq 0$$

Complementarity condition

Adjoint equn.
Design equn.

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design

So from here from the problem statement that we just described we write the Lagrangian we write the Lagrangian which is basically objective function added to that the governing equation with the corresponding Lagrange multiplier function and then Lagrange multiplier for the constraint and multiplied by the volume constraint so we can see what is our F the integrand $pu + \lambda$ times all of this + capital λ this is λ is a function of x where this is corresponding to a functional type constraint or global constraint will be a scalar unknown constant and then the volume constraint.

Here and then we take variation with respect to A that is variation of the Lagrangian with respect to A gives us what we call the design equation this is our design equation as we had said in the last lecture that there is no area of cross section in this equation yet it is a design equation it gives a design equation that is what we will discuss and use it today and this is our adjoint equation and λ is the adjoint variable and it looks similar to our equilibrium equation and hence we can solve for λ there is this equation and this the difference is there is U' here there is λ fine okay .

So we conclude that $\lambda + U$ and then we have the complementarity condition here complementarity condition here which is λ times the inequality constraint either λ is 0 or this expression is 0 in this case we would conclude that λ cannot be 0 because if in that case this one if λ is not 0 then this one need not be 0 then it is strictly less than then we have a problem if λ is 0 then we will

have a problem if λ is not equal to 0 it is fine then this has to be 0 the constraint becomes active okay.

(Refer Slide Time: 05:06)

The slide, titled "Solving the equations", shows the following mathematical steps:

- Equations: $\Lambda - E(\lambda')' = 0$, $(EA\lambda') + p = 0$, and $\Lambda \left(\int_0^L A dx - V^* \right) = 0, \Lambda \geq 0$.
- Derivation: $\Rightarrow (EAu') + p = 0$ leads to $\lambda = u$.
- Optimality criterion: $\Lambda - Eu'^2 = 0 \Rightarrow \frac{Eu'^2}{\Lambda} = 1$.
- Iterative formula: $A(x)^{(k+1)} = A(x)^{(k)} \left(\frac{Eu'^2}{\Lambda} \right)^{(k)}$.
- Handwritten notes: $(k) \rightarrow k^{\text{th}} \text{ iteration}$, $\rightarrow \approx 1$, and $\leftarrow \text{convergence}$.

At the bottom of the slide, it reads: "G. K. Ananthasuresh, IISc" and "Variational Methods in Mechanics and Design".

So solving the equations we first see that these two let me change the color of my ink from these two we get that $\lambda = U$ and then from this one and this one by substituting for $\lambda' = U$ we get this and we get the condition which we call optimality criterion $E u'^2$ by $\lambda = 1$ and this is what we use in the so called optimality criteria method what we do is this is supposed to be 1 so we multiply our $A(x)$ in the (k) iteration here (k) refers to K iteration okay, so what we do is we take an initial guess best initial guess is to take a of X to be a uniform function so that it satisfies the volume constraint you just take V^* divided by L as the area of cross section .

So you satisfy the volume constraint area is the same for all values (x) then you multiply this one what you compute at K^{th} iteration you have to solve the equilibrium equation that is this equation then we will get U and U' then U' goes here anyway you know x modulus will describe how we will compute λ at the k^{th} iteration multiply by A you will get $K+1$ iteration and then keep on repeating it until it converges what do you mean by convergence here that A_{K+1} should be $= A_K$ in other words this quantity over here should be $= 1$ and that is exactly what our

optimality criteria is so convergence is this becomes = 1 that will be our convergence chance until that we have to iterate that so the optimal method works .

(Refer Slide Time: 07:25)

Optimality criteria method

To make it faster (or slower)

$$A(x)^{(k+1)} = A(x)^{(k)} \left(\frac{Eu^{*2}}{\Lambda} \right)^{\beta^{(k)}}$$

$$\int_0^L A dx - V^* = 0 \quad \leftarrow \text{Active constraint}$$

$$\Rightarrow \int_0^L \left\{ A(x)^{(k)} \left(\frac{Eu^{*2}}{\Lambda} \right)^{\beta^{(k)}} \right\} dx - V^* = 0$$

$$\Rightarrow \frac{1}{\Lambda^{\beta^{(k)}}} = \frac{V^*}{\int_0^L \{ A(x)^{(k)} (Eu^{*2})^{\beta^{(k)}} \} dx}$$

$$\Rightarrow \Lambda^{(k)} = \left[\frac{\int_0^L \{ A(x)^{(k)} (Eu^{*2})^{\beta^{(k)}} \} dx}{V^*} \right]^{1/\beta} = \left(\frac{\Lambda}{V^*} \right)^{1/\beta}$$

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design

So to make it faster we just do that but then we introduce a better here are slower depends on what value of β you take we can tune the speed of convergence where you get A_{K+1} based on a A_K the information you raise it to an exponent what should become 1 if you raise it to an exponent it will still be 1 so we choose appropriate value of β we normally choose two or three and that is up to us when you run in the code once we have it we have to find this λ at K^{th} iteration okay, for that we use the volume constraint because volume constraint the complementarity condition.

Where we say λ times constraint = 0 either λ is 0 or constrain 0 if λ is 0 we have a problem that equilibrium question will not be satisfied because you ' it would mean that U' is 0 but U' is 0 then equally equation will not be satisfied with is $EAU' + P = 0$ okay, so that is not possible so lambda should be greater than 0 in that case the constraint should be active which is what we have this is a consequence of active constraint active constraint okay, since its constraint is active we have $ADX \text{ integral} = V \text{ star}$ hence we can take this changed $A(x)$ because when you start

optimization iterative process we take $A(x)$ the initial guess to satisfy the volume constraint now when you update it in this manner.

So this particular one we put for $A(x)$ and try to find this λ that is what we have done we have put this new $1/A_{k+1}$ that is A_{k+1} we integrate over the length 0 to L that should be $= V^*$ that will enable us to solve for this λ raise to β and hence we get this λ at creation in this manner again once you know EA which have assumed to let the optimization start that will give us U_{k+1} and you would have assumed value of β you everything.

You get in the code the denominator we are going to denote by β 1 okay it is not Lagrange multiplier anything which is denoting so that you can take this algorithm and look at the code and understand how the code in mat lab is implemented okay.

(Refer Slide Time: 10:16)

Adding upper and lower bounds on $A(x)$

Min $_{A(x)} MC = \int_0^l pu \, dx$

Subject to

$\lambda(x): (EAu') + p = 0$

$\Lambda: \int_0^l A \, dx - V^* \leq 0$

$\mu_l: A_l - A \leq 0$

$\mu_u: \Lambda - A_u \leq 0$

Design equn.

$\Lambda - Eu'^2 - \underline{\mu_l} + \underline{\mu_u} = 0$

Need not compute.

G. K. Ananthasuresh, IISc
Variational Methods in Mechanics and Design

Now we also discuss another thing which is adding upper and lower bounds on area of cross section area of cross section cannot be very large or we cannot be negative or even close to 0 we wanted to keep it at a min or a L okay lower bound that we choose and then upper bound also that we choose then corresponding multipliers L and this should be U you will be there UL and you U will be there so because of that our design equation will have two extra terms which are indicated here – Min -U_U because AL here we have actually this one should be positive and this should be negative because of the way no.

No sorry I think what we had is aright what we had is right okay, because A has negative here and A has positive - U_U L - U_U = 0 we do not really need to worry about them we do not need to need not compute them unless you are interested in calculating the value of λ analytically okay, wherever your λ value you want to know it is need not compute it will come as part of the calculation as you will see okay, we not complete them we will just go with .

(Refer Slide Time: 11:54)

Handling the upper and lower bounds on $A(x)$

$$\int_0^L A dx - V' = 0 \Rightarrow \sum_1 \left(A \frac{L}{n} \right) + \sum_2 \left(A \frac{L}{n} \right) + \sum_{\text{rest}} \left(A \left(\frac{E u^2}{\Lambda} \right)^{\alpha} \frac{L}{n} \right) - V' = 0$$

① Lower bound ② Upper bound ③ Rest — controlled by the design eqn.

$$\Rightarrow \frac{1}{\Lambda^{\alpha}} = \frac{V' - \sum_1 \left(A \frac{L}{n} \right) - \sum_2 \left(A \frac{L}{n} \right)}{\sum_{\text{rest}} \left(A (E u^2)^{\alpha} \frac{L}{n} \right)}$$

$$\Rightarrow \Lambda^{\alpha} = \left[\frac{\sum_{\text{rest}} \left(A (E u^2)^{\alpha} \frac{L}{n} \right)}{V' - \sum_1 \left(A \frac{L}{n} \right) - \sum_2 \left(A \frac{L}{n} \right)} \right]^{\frac{1}{\alpha}} = \left(\frac{A_0}{V'} \right)^{\frac{1}{\alpha}}$$

denoted in the Matlab code

K. Anurag Kumar, IISc Variational Methods in Mechanics and Design

The idea that the area of cross section integrated over the entire domain 0 to L now we have three portions will have the first portion second portion the third portion first portion is the one that where the area of cross section has reached the lower bound that is where multiplying area by taking it as AL itself the L by n is the DX discretization you see the integration has become discretization because in the computer when implemented we have to discretize this is the upper bound portion upper bound portion and then this is the rest of them which is controlled by the design equation so this is controlled by the design equation okay then you can compute like we did when we did not have lower and upper bounds only thing is in the numerator will have a few extra terms okay ,we have to sum over a set where the area of cross sections have been pushed to lower limit that is if it goes below over limit AL we push it up if it goes above upper limit.

We push it down to AU okay that is the difference and rest of it is the same so just reverse of it and because it is 1 over λ raised to better so we have to take the beet root of this whole thing again in the code this is denoted the is denoted like this in the code in the mat lab code that we are going to see shortly okay, this is our algorithm so if you look at.

(Refer Slide Time: 13:51)

Inner and outer loops

$$\rightarrow A(x)^{(k+1)} = A(x)^{(k)} \left(\frac{Eu'^2}{\Lambda} \right)^{\beta(k)}$$

$$\Lambda^{(k+1)} = \left[\frac{\sum_{\text{the rest}} \left(A(Eu'^2)^\beta \frac{L}{n} \right)}{V^* - \sum_l \left(A_l \frac{L}{n} \right) - \sum_u \left(A_u \frac{L}{n} \right)} \right]^{\nu/\beta} = \left(\frac{\Lambda_l}{V^{*1}} \right)^{\nu/\beta}$$

Outer loop
Updating $A(x)^{(k)}$

Inner loop
Updating $\Lambda^{(k/j)}$

G. K. Ananthasuresh, IISc
Variational Methods in Mechanics and Design

How the algorithm works we have $A(x)^{(k+1)}$ being updated from the k^{th} iteration information of x^k and then we keep on looping this is our outer loop to update $A(x)$ when x^k and $A(x)^{k+1}$ become = meaning this what multiplies it becomes = unity as we have said that we have converged right but an inner loop is needed that inner loop to update the λ so we have another index J here to update λ why do you need update λ because.

When you update $A(x)^{k+1}$ in this manner area in some portions can exceed the lower bound or upper bound then you have to push up in the case of lower bound being exceeded or push down in the case of lower upper bound exceeded so when you do that the λ value has to be updated because λ was calculated in the iteration of the inner loop when you begin assuming that nothing has gone above and below the lower bounds now we are pushing it we have disturbed the area of profile.

So you have to recompute the value of λ which is what necessitates this inner loop so we have an outer loop inner loop as indicated here the outer loop you need to update area of cross section that needs the finding of the λ^k at the k^{th} iteration for that you run inner loop and how when do we stop we stop when no more elements exceed the upper and lower bounds which ever exceed we push them to the either lower upper bound as the case may be but we keep on doing it until there is no change.

Then we stop the inner loop so there is an outer loop that goes repeatedly with the index K there is an inner loop that goes with the index J so that is what is going to happen so for every outer loop there will be a few inner loop iterations and then exits goes to outer loop and outer loop converges when this quantity becomes equal to 1 and then a $k + 1$ gets very close to AK based on the tolerance that we specify in the code so let us look at the mat lab code.

(Refer Slide Time: 16:23)

Matlab code

```
%%  
% Stiffest bar for given volume and axial load  
% Solved using the optimality criteria method  
% Code written by Gaurav Nair in 2011, IISc-Bangalore  
% Commented and modified thereafter for NPTEL MOOC Course  
%%  
close all; clear variables; clc; hold off;  
%% Data : bar  
L = 10; % length of the bar  
%% Data : FEA  
n = 100; % number of elements  
nNodes = n+1; % number of nodes
```

Handwritten annotations:
- "House-keeping commands" points to the `close all; clear variables; clc; hold off;` line.
- "elements" points to the `n = 100;` line.
- "nodes" points to the `nNodes = n+1;` line.
- A diagram shows a horizontal line with four nodes labeled 1, 2, 3 and three elements labeled 1, 2, 3.

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and...

Now so this is those of all of you should be aware of mat lab by now otherwise you can go to math works website and look at tutorial files okay a percentage sign in mat lab shows a comment line that will not be executed by mat lab software so we can write your name and things like that so for example this code we are modified it for NPTEL massive online open course so course is redundant here okay.

So there are some housekeeping commands meaning that we have to do close all figures that are there clear all variables clears screen and hold off all these are basically what I call housekeeping commands you have to keep your you know code in it and it is workspace everything we have to keep it clean these are housekeeping commands it is a good idea to do add these lines any time

you write mat lab code we are specifying length here and the comments here say what it is and data for finite element analysis.

Because that is what we will use here we are choosing 100 along the length of the bar and number of nodes in that case will be one more than the number of elements number of elements is n which is 100 number of nodes will be one more because if I say I have two elements how to have two three nodes this is 12 these are elements if it is a bar just a straight bar so we will have one more than the number of elements for the number of nodes okay that is straightforward enough and you can change the number and play with the code.

(Refer Slide Time: 18:22)

Matlab code

```

l = (L/n)*ones(n,1); % vector containing length of each element
A = 6*ones(n,1); % vector containing initial area of each element
E = 210*ones(n,1); % vector containing elastic modulus of each element
nx = zeros(n,1); % vector initialization
for i=1:n
    nx(i+1) = nx(i)+l(i); % x co-ordinates of the nodes along the axis of
end % the bar
ny = zeros(n+1,1); % y co-ordinates of the nodes along the axis of the
% bar

```

Handwritten annotations: $n \times 1$ vector (circled), $2, x_2, x_3, \dots, x_{n+1}$ (with a bar diagram above it).

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design

Now after that we define the length vector okay the once n / l that means that it is a n / l array or vector okay and that is magnitude is l / n so we have n elements and length of the bar is l each element will be of length L divided by number of elements so L / n and then we have this vector containing initially area here we are just taking constant 6 ideally we can take volume that we are going to specify be v^* divided by number of elements but does not matter it just starts from some uniform guess we can change the number in the code and we have the young's modulus each of them is given as n / l vector okay.

Because that is how our finite element code which is part of this code is implemented and then we have the x coordinates for this it is a one-dimensional problem so it is like this no different nodes are there we are to the x coordinate x_1 x_2 x_3 of course until X we have $n + 1$ nodes so that is x_{n+1} and then y we do not have any of them they are just 0 it is a one-dimensional problem which zeros okay.

(Refer Slide Time: 19:48)

Matlab code

```

% F = 20*ones(n+1,1); % uniformly distributed force
% F(n/2,1)=50; % force at the center
% F(n+1,1)=100; % force at the end

ncon = zeros(n,2); % vector initialization
for i=1:n
    ncon(i,1)=i; % connectivity vector of the bar
    ncon(i,2)=i+1;
end

% boundary conditions for fixed-free condition
% dispID=[1]; dispVal=[0]

% boundary conditions for fixed-fixed condition
dispID=[1; n+1]; dispVal=[0;0]

% boundary conditions for free-fixed condition
% dispID=[n+1]; dispVal=[0]

```

Handwritten notes on the slide include: $p(x) = p_0 = 20$, $F_{(n+1) \times 1}$, and a diagram of a bar with nodes 32, 33, and 34. The left end is labeled 'fixed' and the right end is labeled 'free'.

C. K. Ananthaswamy, IISc Variational Methods in Mechanics

Going to the next few lines in the code we have copied the lines over here to explain then we will go to mat lab code in math lab soft and run it and we are taking the force here if you look at these commands there are these comments here so whether you want to put uniformly distributed force the $P(X)$ is like p_0 which in the steak is taken as 20 units then you would give this 20 times once $n + 1 / 1$ because number of nodes are $n + 1$ so you have to do that so f vector that we have will be $n + 1 / 1$ okay if you want force at the center you have to know $n / 2$ 1 you will specify some force or force at the end f_{n+1} times 1 okay.

So $n + 1$ times 1 you have the for specified at the n other time you put a percentage sign here and remove this percentage becomes midpoint then you put back percentage remove this force in the

end basically you can specify whatever force you want algorithm is going to work okay and then there is as connectivity to say which nodes make up an element so if I have a bar and several nodes here let us say this is 32, 33, 34 then 32 and 33 makeup.

Let us say 30 second element that is what is connectivity does and then most important thing for us the boundary conditions if you see we have bound to connect four fixed free then this π done that is displaced upon the degree of freedom ID displacement ID for the first node value is zero that is fixed and free that is we have a bar this end is fixed that it is free so this is fixed and free okay.

Similarly fixed then you put 1 and $n + 1$ both are specific to be 0 and then free fixed we have $n + 1$ value is 0 what value this value is what value you are specifying for that node all these are self-explanatory we are going by line by line here.

(Refer Slide Time: 22:11)

Matlab code

```

%% Data : optimality criteria
tol = 1e-3; % tolerance to check convergence
Its=50; % maximum number of iterations
eta=.3;
vStar = 20; % upper bound on volume
Amax = 6; % upper bound on area
Amin = 1; % lower bound on area
historyA=zeros(n,Its+1); % initialization
historyA(:,1)=A(:,1); % matrix to store values of area after every
% iteration
  
```

$(1)^\beta$

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and

And then once you have it we also have to give that tolerance that is tolerance is given as 10^{-3} here let us up to you can increase it decrease it that is going to decide when area of cross section A_{k+1} how close it is to A_k when it quits that is what we have and in this etta our

data okay that is this in the code it is written as beta actually better in when I explained it that is we had something that event will be equal to 1 we were raised into beta right that is this better.

Oh here it is taken as point3 we are slowing it down rather than speeding out that is something that you have to decide when you run and then v^* is 20 units you can put whatever these numbers here and AI and AU are here this is AU upper bound and AL this lower bound and then history a basically it stores everything you can see how the bar we started a uniform bar how the shape changes we can see the figure file to see how optimization is being carried out and thus those commands here.

(Refer Slide Time: 23:38)

```

Matlab code
%% Optimize criteria implementation
updateProc(A,d,ro,r) % get initial area

for i=1:20
    [u,Status,PKing,GC] = fembar(A, L, nE, nN, rocon, rHNode, f, dispC, dispA);
    % FEA gives back displacement u corresponding
    % to present area, keeping area boundary
    % conditions
    for a=1:n
        uDeriv(a)=(u(a+1)-u(a))/L; % compute derivative of u using finite
        % difference
    end
    uDeriv(1)=0;
    for j=1:n
        uNode(j)=uNode(j)-A(j)*(r(j)-uDeriv(j))^2/ro; % compute uNode
        % from design
        % equation
        uNode(n)=uNode(n)+ro*(r(n)-uDeriv(n)); % add volume
        % constraint
    end
    for j=1:n
        A(j)=A(j)*(r(j)+uDeriv(j))^2/rocon/ro; % update area
        % using design
        % equation
    end
end
  
```

And this is the real implementation the font is a little small here but since you have the code that is also posted along with this we will be able to see that more clearly or pass the screen and look at it so first thing is we are calling fem bar that takes the area of cross section moment second moment this actually is not needed here this is for a this is actually length of the bar L here it is not I a length of the bar and then X model is x coordinates y coordinates are not needed encon number of elements number of nodes force vector de Spidey dispel that we describe all this is input to finite element code that is going to return you displacement and a little bit of other stuff that we do not need right.

Now for this discussion it returns reactions and P internal forces and K singular matrix before displacements are applied and AC the strain energy we are interested in U we use that you to compute u' using finite difference method so this one is finite difference method to compute it is all numerical now we are computing finite difference using finite difference method and then we initiate this $\lambda = 1$ which was the numerator of our update scheme that we had pointed out in the slides earlier let us go there so we can look at that quickly.

(Refer Slide Time: 25:05)

Matlab code

```

%% Data : optimality criteria
tol = 1e-3; % tolerance to check convergence
its=50; % maximum number of iterations
vStar = 20; % upper bound on volume
Amax = 6; % upper bound on area
Amin = 1; % lower bound on area
historyA=zeros(n,its+1); % initialization
historyA(:,1)=A; % matrix to store values of area after every iteration

```

β
 $\begin{pmatrix} 3 \\ 1 \end{pmatrix}$

(Refer Slide Time: 25:06)

Matlab code

```

F = 20*ones(n+1,1); % uniformly distributed force
F(n/2,1)=50; % force at the center
F(n+1,1)=100; % force at the end

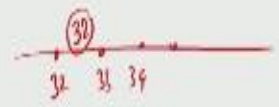
```

$p(x) = p_0 = 20$ F
 $(n+1) \times 1$

```

ncon = zeros(n,2); % vector initialization
for i=1:n
    ncon(i,1)=i; % connectivity vector of the bar
    ncon(i,2)=i+1;
end

```



```

% boundary conditions for fixed-free condition
dispID=[1]; dispVal=[0];

```



```

% boundary conditions for fixed-fixed condition
dispID=[1; n+1]; dispVal=[0;0];

```

```

% boundary conditions for free-fixed condition
dispID=[n+1]; dispVal=[0];

```

(Refer Slide Time: 25:07)

Matlab code

```
%S  
% Stiffest bar for given volume and axial load  
% Solved using the optimality criteria method  
% Code written by Gaurav Nair in 2011, IISc-Bangalore  
% Commented and modified thereafter for NPTEL MOOC Course  
%S  
close all; clear variables, etc; hold off;  
%% Data : bar  
L = 10; % length of the bar  
%% Data : FEA  
n = 100; % number of elements  
nNodes = n+1; % number of nodes
```



(Refer Slide Time: 25:08)

Inner and outer loops

$$\rightarrow A(x)^{(k+1)} = A(x)^{(k)} \left(\frac{Eu'^2}{\Lambda} \right)^{\beta^{(k)}}$$

$$\Lambda^{(k+1)} = \frac{\sum_{i=1}^{n_{el}} \left(A_i E_i u_i^2 \right)^{\beta} \frac{L_i}{n_i}}{V^{\beta} - \sum_i \left(A_i \frac{L_i}{n_i} \right) - \sum_s \left(A_s \frac{L_s}{n_s} \right)} = \left(\frac{\Lambda}{V^{\beta}} \right)^{\beta}$$

The diagram shows the iterative process. The outer loop, labeled 'Outer loop', updates $A(x)^{(k)}$. The inner loop, labeled 'Inner loop', updates $\Lambda^{(k)}$. The equations show that $A(x)^{(k+1)}$ is calculated using $A(x)^{(k)}$ and $\Lambda^{(k)}$. The inner loop calculates $\Lambda^{(k+1)}$ based on the current $A(x)^{(k)}$ and $\Lambda^{(k)}$. The process repeats until convergence.

So we said that this in the code is denoted as.

(Refer Slide Time: 25:10)

Adding upper and lower bounds on $A(x)$

$$\text{Min}_{u(x)} MC = \int_0^l pu \, dx$$

Subject to

$$\lambda(x): (EAu') + p = 0$$

$$\Lambda: \int_0^l A \, dx - V^* \leq 0$$

$$\mu_l: \underbrace{A}_l - A \leq 0$$

$$\mu_u: A - \underbrace{A}_u \leq 0$$

Design equn

$$\Lambda - Eu'^2 - \mu_l + \mu_u = 0$$

Need not compute.

(Refer Slide Time: 25:12)

Optimality criteria method

To make it faster
(or slower)

$$A(x)^{(k+1)} = A(x)^{(k)} \left(\frac{Eu'^2}{\Lambda} \right)^{\beta^{(k)}}$$

$$\int_0^1 A dx - V^* = 0 \quad \leftarrow \text{Active constraint}$$

$$\Rightarrow \int_0^1 \left\{ A(x)^{(k)} \left(\frac{Eu'^2}{\Lambda} \right)^{\beta^{(k)}} \right\} dx - V^* = 0$$

$$\Rightarrow \frac{1}{\Lambda^{\beta^{(k)}}} = \frac{V^*}{\int_0^1 \left\{ A(x)^{(k)} (Eu'^2)^{\beta^{(k)}} \right\} dx}$$

$$\Rightarrow \Lambda^{(k)} = \left[\frac{\int_0^1 \left\{ A(x)^{(k)} (Eu'^2)^{\beta^{(k)}} \right\} dx}{V^*} \right]^{1/\beta} = \left(\frac{\Lambda}{V^*} \right)^{1/\beta}$$

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design

(Refer Slide Time: 25:13)

Matlab code

```

%% Optimally criteria implementation
updatePos(A,d,xx,1) % just initial area
for i=1:n
    [r,disp,PKing,DC] = fmincon(A, [C, n0, n0], r0, n0, r0, n0, f, disp0, disp, w);
    % FEA (give back displacement) corresponding
    % to present area, listing are boundary
    % conditions
    for a=1:n
        uDis(a) = (u(a+1) - u(a))/h(a); % compute derivative of u using finite
        % difference
    end
    lambda = 0;
    for j=1:n
        lambda = lambda + A(j)^2 * (r0(j))^2 * uDis(a)^2 / w(a); % compute lambda
        % from design
    end
    lambda = lambda / w(a)^2 * V(a); % and volume
    % constant
    for j=1:n
        A(j) = A(j) * (r0(j))^2 * lambda / w(a); % update area
        % using design
    end
end

```

Handwritten annotations:

- Outer loop at i* (circled 'i' in code)
- Finite difference method* (pointing to derivative calculation)
- eta* and *β in the slides* (pointing to lambda calculation)
- length* and *input* (pointing to A and d)

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design 11

λ 1 okay that is what we are talking about that is all of this stuff is λ 1 okay that is there in the code so we are here λ 1 and that summation that we do if you remember that that had a summation that is what we do and then once you have λ 1 you compute λ is λ 1 way V^* raised to 1 over β as I said data in the code is better in the slides which is a symbols okay but different and then you update a j equal to a_j that is k^{th} information and what should be equal to 1 okay.

That is how we update and there is a iterative loop you see this number of iterations so iteration outer loop starts here outer loop starts here okay.

(Refer Slide Time: 26:14)

Matlab code

```
flag=false;
for j=1:n
    if (A(j)>Amax)
        A(j)=Amax; % all areas greater than Amax are set to Amax
        flag=true;
    end
    if (A(j)<Amin)
        A(j)=Amin; % all areas less than Amin are set to Amin
        flag=true;
    end
end
```

And then we are checking by setting a false for the flag and then checking whether it has reached a max or a min if it is there we set it to e masts are a min if it exceeds upper bound we set it to a max with exceeds lower bound set it to a min al and au and this is a u and this is al.

(Refer Slide Time: 26:44)

Matlab code

```

% start of inner loop to make sure that updated areas are within upper
% and lower bounds
while flag
    % inner loop begins here
    % update remaining areas using design
    equation
    A_j = A_j * (1 - 2 * lambda) * eta;
    and
end
flag = false;
for j = 1:n
    if (A_j > A_max)
        A_j = A_max; % all areas greater than A_max are set to
    else
        if (A_j < A_min)
            A_j = A_min; % all areas lesser than A_min are set to
        else
            flag = true;
        end
    end
end
updateP(A,d,n); % get area
end

```

G. K. Ananthasuresh, IISc Variational Methods in Mechanics and Design

We do that then the inverse now so inner loop begin here okay in the loop that we discussed in the slide you should go through it carefully inner loop begins here that goes where you check whether it is exceeded or not when it does not exceed you update using our usual design equation otherwise you set it to a max and min and then check first when you come if anything has gone you say you assign it and you are up count that is how many elements are there above which have reached MX and then how many have reached the lower bound amen or a you one AF you keep count of that.

And then use that when you are updating for λ and that is over here in terms of computing this λ okay now you have to take that λ one where you are subtracting that again it correspond to the formula we had in the slide so up count and down court we count how many elements are have exceeded upper bound and hence are pushed down to upper bound how many have exceeded lower bond and then pushed up to lower bound.

So all this we had subtract from v-star remaining one of them we have used to compute our λ so go back and look at the slide so let us do that quickly to see what we are talking about so we are talking about this.

(Refer Slide Time: 28:21)

Handling the upper and lower bounds on $A(x)$

$\int_0^L A dx - V^* = 0 \Rightarrow \sum \left(\frac{A_i L}{n} \right) + \sum \left(\frac{A_i L}{n} \right) + \sum_{\text{over sum}} \left(A \left(\frac{Ew^2}{\Lambda} \right)^u \frac{L}{n} \right) - V^* = 0$

① lower bound ② upper bound ③ Part — controlled by the design eqn.

$$\Rightarrow \frac{1}{\Lambda^{(k)}} = \frac{V^* - \sum \left(\frac{A_i L}{n} \right) - \sum \left(\frac{A_i L}{n} \right)}{\sum_{\text{over sum}} \left(A \left(\frac{Ew^2}{\Lambda} \right)^u \frac{L}{n} \right)}$$

$$\Rightarrow \Lambda^{(k)} = \left[\frac{\sum_{\text{over sum}} \left(A \left(\frac{Ew^2}{\Lambda} \right)^u \frac{L}{n} \right)}{V^* - \sum \left(\frac{A_i L}{n} \right) - \sum \left(\frac{A_i L}{n} \right)} \right]^{1/u} = \left(\frac{\Lambda}{V^*} \right)^{1/u}$$

denoted in the Matlab code

G. K. Ananthasuresh, 75c Variational Methods in Mechanics and Design

So how many have exceeded lower bound and upper bound that is what we are subtracting from v^* if you see the code we are subtracting this and this okay so once we do that we go through it and then finish the inner loop you can see the text here but this is inner loop ends here inner loop ends here so it began here it ends here okay we are basically taking care of computing λ in the new loop and then it is inside the outer loop and then finally.

(Refer Slide Time: 29:05)

Matlab code

```
historyA(:,i+1)=A(:,i); % store A at the end of each iteration to  
% historyA  
  
tolcheck=max(abs(historyA(:,i+1)-historyA(:,i))); %  
if tolcheck<=tol % check and stop iterations if solution has  
break; % converged  
end  
  
updatePlot(A,d,nx,n) % plot area  
end
```

We store everything check for convergence here is where we are checking for convergence to see how first of all we are storing all the history so we can update plots we can see how optimization is actually working and that said.

Now what we will do is we will go to math lab software and look at the actual running of the code so we go here I will start mat lab over here which I have already started and the code bar optimization is over here that you will see on the screen in a minute so all the code that I showed you in the slides is exactly listed here okay so you can compare with the slides and here and the data you can change whatever and do it okay what we will do is now we will look at the data 10 units.

And this is only for rendering we have diameter otherwise it is not needed just to show it in the graphics number of elements is 100 we can change to whatever we want number of nodes as we already said is n number of elements + 1 and we have six for the uniform initial young's modulus to 10 it is actually giga Pascal but does not matter for looking at the shape when you solve an actual problem you put their actual numbers with the correct units and so forth and we went through all of this let us just look at what boundary conditions are boundary conditions only the first node is fixed at zero value meaning we are looking at a fixed free condition okay and what is the force the force commands are up here we are using uniform distributed force if you

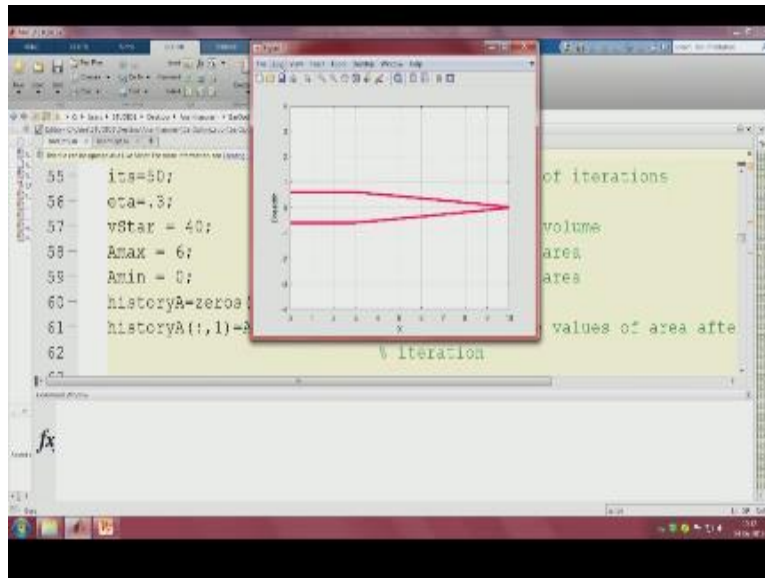
want other ones they are all over here we can remove the comment and put them that is this one is for midpoint load only this is for n point load only right.

Now we have uniformly distributed load force $P(x)$ equal to P_0 is 20 units here and then we have fixed free bar let us run this and then see what happens you will see that how the thing change change too quickly is converging already okay and the cursor comes back over this window that means that we have converged let me look at the figure window it is done so just so that you will not miss it I am going to run again watch it this time how it goes iteration wise and when the command prompt comes back now it has converged it come to this very quickly and you see that it has reached the upper bound over here that we have set okay.

And they reach the lower bound we have set let us find out what those values are a max and A max somewhere so a max is 6 a min is one okay let us look at the figure so it is a max is 6 so I think the way it is done yeah I think it is the drawing here does not really show that a minima because some scaling must have been done over here okay so but basically has A max and there is any mean we can check and see why it is showing breadth of the beam is so much but here the nature of the thing is important it is linearly profile linear profile here which is a lower limit and upper limit okay.

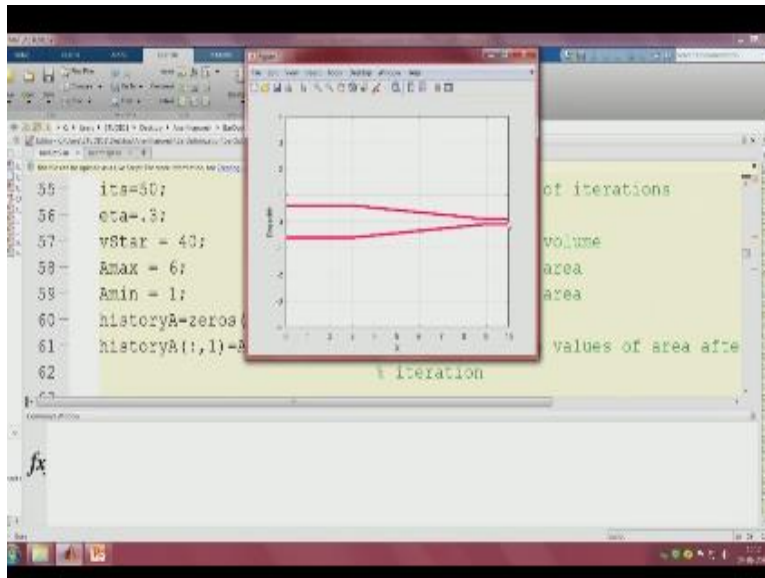
If you change the numbers it should change actually but here we are taking from here to here looks like there is a scale factor of some kind in order to show here it is only rendering but actually area profile will be calculated okay.

(Refer Slide Time: 33:14)



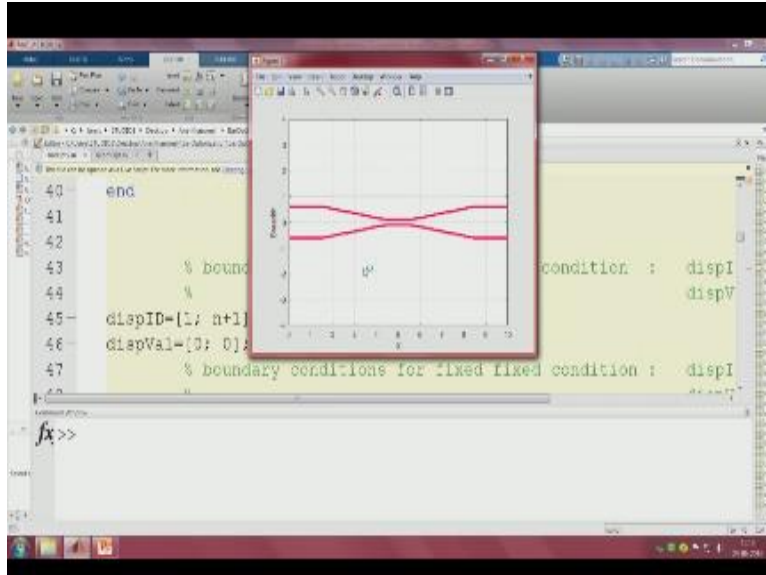
Now what we will do is we will well let us let us try this if a min is actually working let me put this as 0 and then run to see if it works yeah it did work so the tip point went to zero so there is scaling scale factor does not matter in rendering now there is a scale factor remains 0 even after scale you should come to zero became a sharp one here okay a men and a max some value okay.

(Refer Slide Time: 33:47)



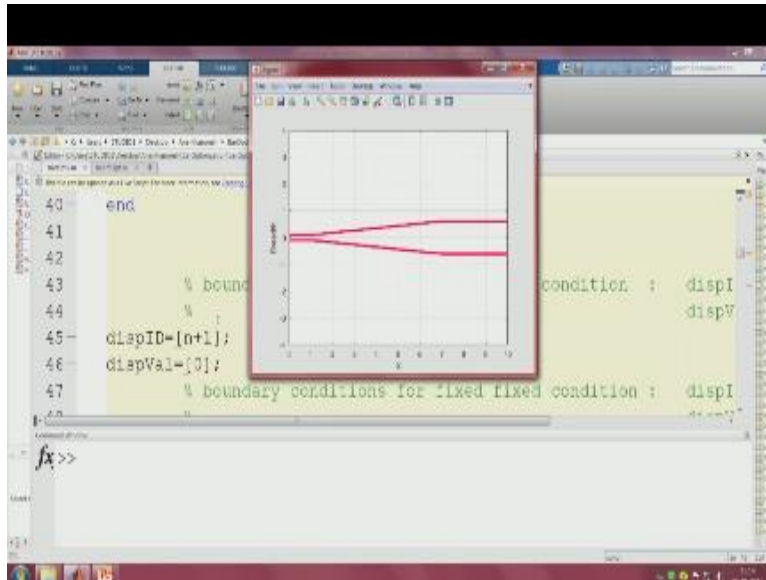
Now what we will do is we will put back this again to let us say one or something okay now it we run it and it is going to go like that it lodged it is not allowed to go to zero right it is still okay I would Stan so now we are going to change the displacement boundary conditions so it was a fixed free.

(Refer Slide Time: 34:21)



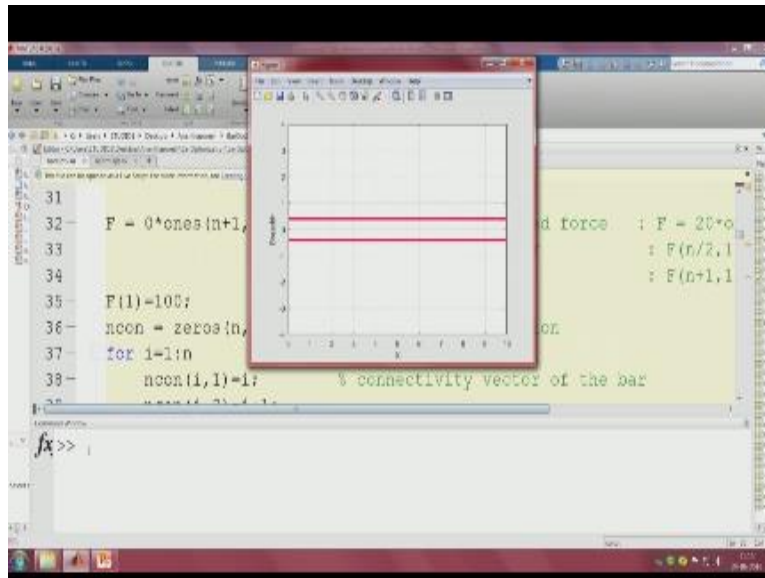
Let us make it fixed, fixed so what I will do here is to fix also the other end that is $n + 1$ and I need to add the value because I am fixing it that should also be 0 should be fixed, fixed bar now okay comma r this either one is fine destroy comma so this has not changed one second I think we have to put the this one to be semi colon here this value also is semi colon okay, okay now you see it has changed we have to put semicolon here 1 and $n + 1$ node and the 00 value fixed, fixed this fix is tapering down and tapering up so fixed, fixed one is different.

(Refer Slide Time: 35:16)



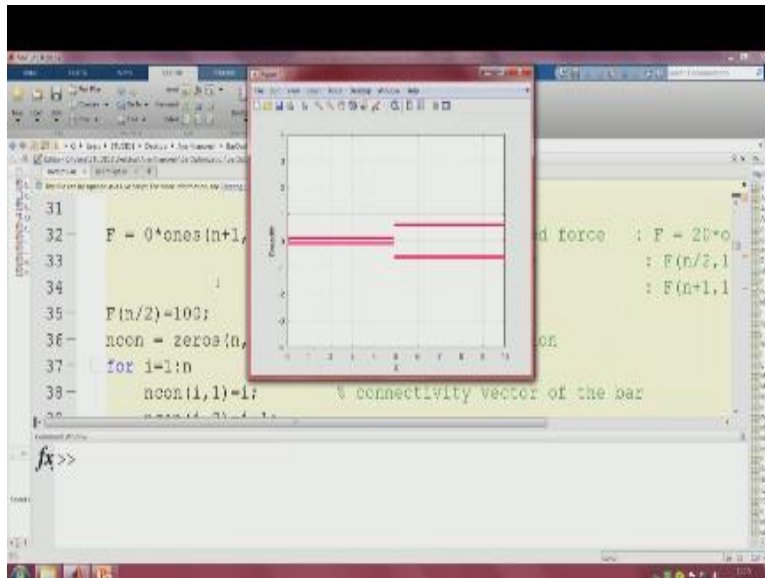
We can also do fixed free, free fixed so I will remove this so only $n + 1$ the first node is free the last node is fixed if we do that it should show a mirror image of that so previously it was tapering that way and making this lower bound then it becomes lower bonded because upper bound basically the code works whichever way we want okay and now if you were to play with the forces so we had different forces here.

(Refer Slide Time: 35:44)



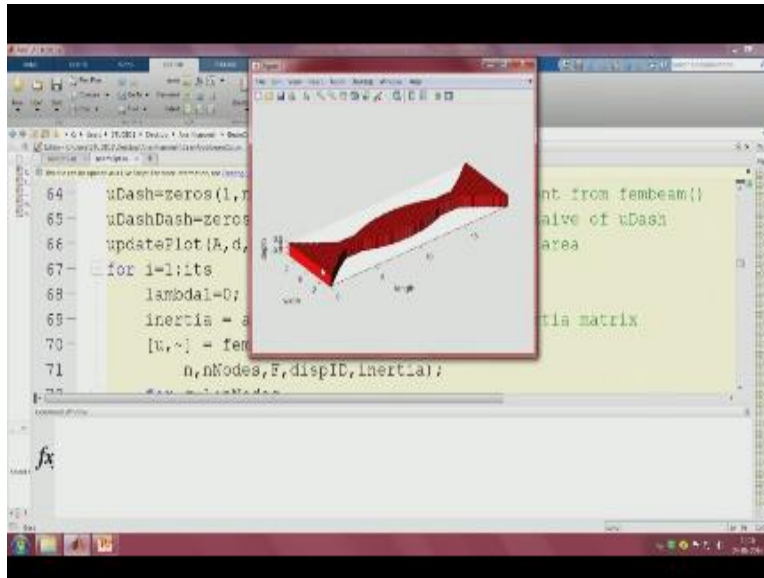
So if I want to apply force only at the end okay let us take that and I will make it zero right that that array is needed and then I will pray that only n plus 1 at the end I would like to put 100 everywhere else it is zero okay the profile if you see the profile is going to obtain profile fixed free uniform loading this is the one which you can verify analytical thing we have done in the earlier in an earlier lecture now if I put constant loading okay at not n because that we have fixed that the first node okay let us see what happens then it remains constant it just adjust to the volume and that said it is converged already because the prompt came in so I will run again to see very quickly it converges and comes over fixed free but when you put this loading here again at one node so we can change it we can add another node if you want.

(Refer Slide Time: 36:49)



Let us say I added at the NY to the mid one somewhere in the midpoint let us see then you see it goes to a min here is fixed over here and free when I whenever a load is only midway I do not need material over here the algorithm realizes that many times when you implement algorithm like this it will not miss the intuition that you have and that you have to check various cases like we have doing now so we can basically run everything all right so having seen the barcode I would like to show you the beam code also.

(Refer Slide Time: 37:25)



Which we have discussed optimal criteria first affair structure so we will run this beam code now okay again you can go through all these lines which basically mirrors the bar code but for the case of beam where there will be fe m beam that will be called okay that we can see instead of bar now we will have the beam code being run so you have a fee m beam that also will be given in the website for you to download to run the code yourself okay let us run it whenever you run massive asks for change in the folder which we are doing here so it is also you can see how it changes looking at it we can say it is a fixed, fixed beam.

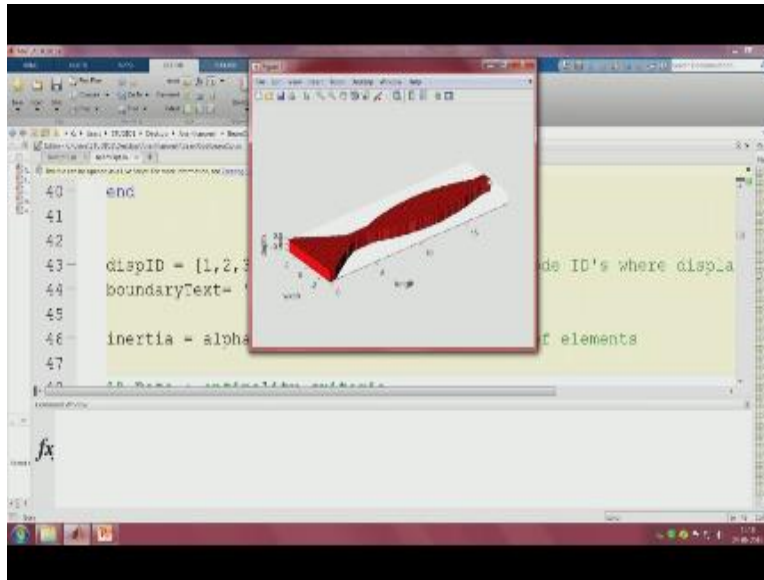
It is still some small changes are still happening now it is not straight line anymore like in the bar in the case of abeam this is for uniform transverse loading that is acting perpendicular to the beam here and now until the command prompt comes here it is not done so it is still changing slightly you can see some lines here and there slightly flickering means that it adjusting fine-tuning accord to the tolerance that is specified in this program.

(Refer Slide Time: 38:48)

```
40 end
41
42
43 dispID = [1,2,3, 3*nNodes-2, 3*nNodes-1, 3*nNodes]; %node ID's w/n
44 boundaryText= 'fixed-free';
45
46 inertia = alpha*A; %inertia of elements
47
```

Now it has come back so it is done we have fixed, fixed beam so we can go back and change the boundary conditions so if you see in the beam code we have one two three, three times number of nodes minus 2, three times number of nodes minus one, three times number of nodes basically every node will have three degrees of freedom X displacement Y displacement and then rotation the slope okay so by specifying one two three I am fixing the beam like cantilever at one end one two three that is the it cannot move in the X Direction it cannot move in y direction it cannot rotate also and same thing I am doing in the right hand side because three star n nodes 3 star n nodes minus 1 3 2 n nodes minus 2 all are fixed okay.

(Refer Slide Time: 39:36)



If you just want to see what happens if I remove this here when I remove that it is going to become like a pin joint because I am allowing to rotate then remember the profile here it is symmetrical because fixed, fixed beam now by removing the three star n nodes I made it like a pin joint so let me run this you will see that it changes the pin joint actually wants it wants no material there if I make area of min here equal to zero will actually become a point right.

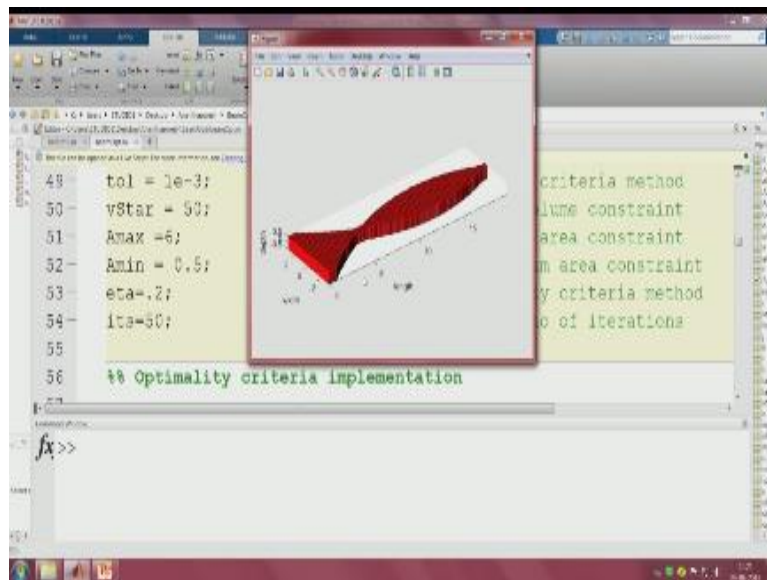
Now it is going to a min some value that is specified okay, it is still running because the command prompt has not comeback when it comes we will go there is some small changes you cannot see, see how quickly the Appalachia algorithm converges we are only being overly careful by giving a very small tolerance value and it takes some while to put but you do not see any change but there is a little change happening.

So i want to show you that if I make a min equal to 0 what happens let us try that a min over here it is one let us make it 0 okay, so make it 0 yeah it is going to 0 so it is becoming a sharp tip and since over here also tipping sharp tip now it is actually deciding to go apart and come together go apart come to that is what iteration is happening okay, transverse loading is there instead of becoming two beams it is just adjusting a finally it went because you cannot have 0 because was going like this so this may take a while for it to resolve number of maximum iterations you put it

will be basically it wants to make it once come down we are allowing it to go to 0 it has gone to 0 and goes there looks like a fish, okay.

So transverse beam the loading acting and a fish is you know we can say optimal but that is not correct it will just looks like a fish it has now come back right, max new iterations is 50.

(Refer Slide Time: 41:43)



So instead of putting 0 let us put 0.5 or something like that then we will see the effect of that already, okay. Now it has some area of cross section it's not going to 0 this is for fixed and pinned beam okay, when it comes back then it is done we will get to command prompt, prompt and beam code would have been done okay, so this way we can change the boundary conditions we can change loading and look at the solutions optimal solutions for the profile okay, very small tiny changes must be happening here that is why it is sticking so there is some flickering here, here there okay.

So it is just running until now it is done command prompt has comeback so what we will do now going back to the this going to let us make it like a cantilever I will remove these are the degrees of freedom completely make the other end free only 1,2,3 left node fixed in xy and the slope, so

if you do that then we should see that part it is coming down like a curve in the case of a bar to a straight line but here it is not and it just goes to a min the other part okay, so you should learn by learning it several times by looking at it you have to guess the boundary condition it is fixed and free, okay.

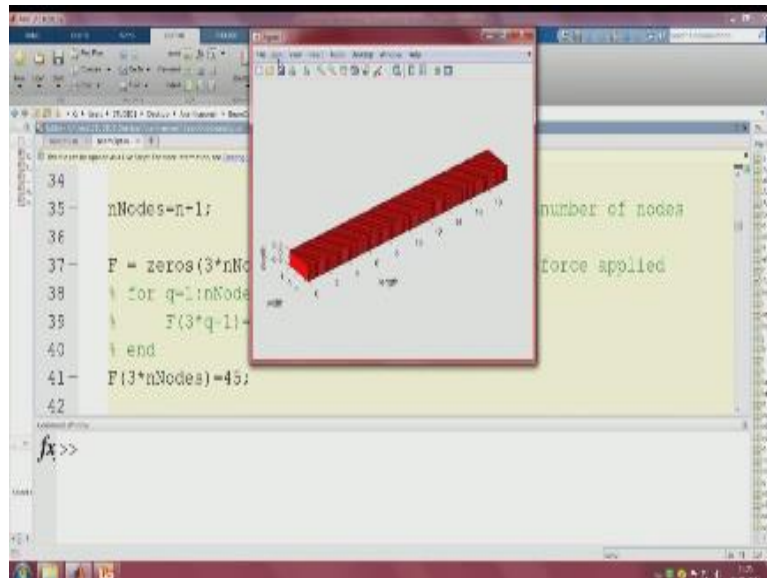
We can try lots of different boundary conditions fixed and guided and many others okay, fixed and guided means that we fix it okay, let me back so roll said, so that will make it 1 okay so going to this displacement boundary conditions let it go yeah, over here so fixed guided means that I will fix that the last degree of freedom that is slope degree of freedom and I will fix the x thing but y are alone meaning that I will have this one the right-hand side he is able to move up and down but cannot rotate this is not alone but can move up and down.

So I have a beam this right-hand side cannot rotate but it can move up and down such a condition is what we have specified it will give you a slightly different one it almost looks like fixed, fixed but it is allowed to move there okay that is what we had put, right. So it looks slightly asymmetry now slightly and we can change the boundary condition and look at what happens okay, X displacement y displacement whichever way you want we can try all of those okay, it is still running there are fine adjustments being done okay, so I would encourage you to try out these codes because we will be giving a programming assignment also for you so that you can try these programs and look at your own optimal profiles and run the programs.

In the future lecture we will have a strong guest column design which again can be done using optimality criteria method. Once you follow the algorithm that we have developed by writing down a Lagrange equations and solving the joint equation analytically and get the optimality condition from the design equation and developing that kinetic criteria method implementation is quite easy in fact once you have finite element analysis doing structure optimization is not at all difficult it is very straightforward.

The code may look long but it is very clear and it is very easy and this lecture we compared the code with algorithm so you can understand the code that is given okay, so we can run lots of them here.

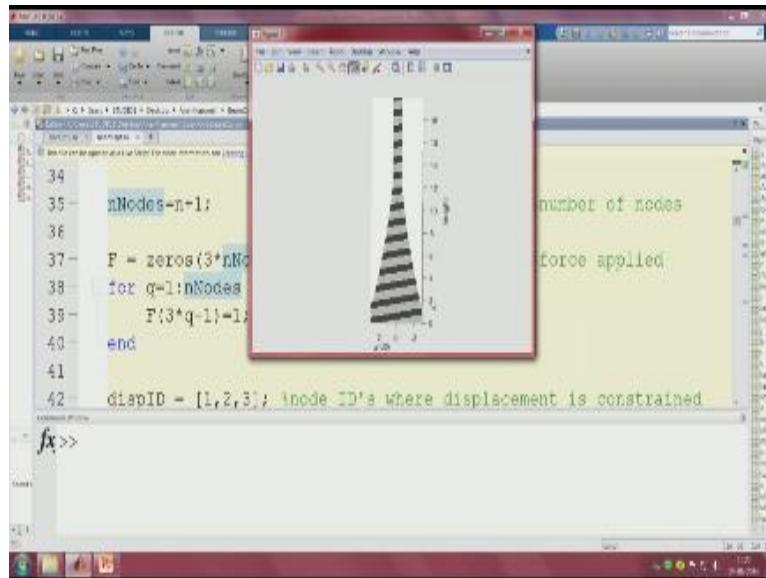
(Refer Slide Time: 46:02)



So I can again go back to cantilever let us see how it looks when I change the loading to only point loading so let us see where the force is forces here so now it is force applied on all of them like one okay, so we can make it uniform distributed or only at one node now it is going 1 to 1 nodes all of them it is giving but we can make only the last one have lost one has a movement anything that we can try because there are so many so once everything is 0 let me comment this line let me comment this line, okay and give force okay that is $3*n$ nodes number of nodes that is the last degree of freedom that is I am applying only a moment load.

Let me put some 45 does not matter what it is okay, if we do that loading let us see the code works or blows up so we have constant movement when there is constant movement we got conservation of cross section that makes sense if you go back and look at analytical thing if constant movement only movement we applied for a cantilever at the end then the bending moment is the same throughout the beam and hence up my criteria method would tell you that area of cross section because we assumed second moment of area equal to area of cross section some $\alpha d^2/12$ that is just an $\alpha\beta=1$. Then what we get is uniform cross section that is what it is it converge you very quickly.

(Refer Slide Time: 47:47)



Instead if I apply a vertical load okay, that degree of freedom will be one less than that okay, this becomes a transverse load let us see what happens yeah, now these the tapering thing that we have so it in this case has not reached the upper limit but has reached the lower limit okay, it is still running because command prompt has not come back now it has comeback we can rotate and then see how it looks now it is a profile that is straight goes like this, okay.

So yeah, that is how it is all right so if you want again the uniform distributed load you can have that so we can move back and remove uncomment these things no comment run it again for a cantilever it will go back to that they will be slight different now it's a little bit curved we are a cantilever was when you have only a transverse load at the free end of a cantilever becomes straight linear but if you be distributed load it gets a little curved and you can see this now yeah, a little curve compared to the one that is there.

In fact one may say that Eiffel Tower if you remember that has wind load which we assume that it is constant then it will have a shape so look at this shape does it not remind you of that Eiffel Tower because there is a M input if there is no M in it just goes like a spire I can make $n_m = 0$ it

will go there we will just do that the last one that we tried a min was somewhere in the code here yeah, a min let us make it 0 okay, then it will just go like a point thing.

So you can already see it is happening okay, once you allow 0 it just goes there with the load that is being there we will wait for the prompt to come up so it still making some small T is going to a max and then nm okay, and that is 0 so this became a pointed thing like spires. So many of these structures that are built were also optimal they probably did not run all these algorithms but the intuition guides good structural engineers to come up with optimal solutions, but you have algorithms to get that now.

So let me just turn it around and see how like a church spire coming here with a pointy tip alright, so you have the coding hands now you can run it and understand how it works and enjoy the code, bye.