FEM2021/11/16

Tuesday, November 16, 2021 2:50 PM

Step 10: calculating stiffness matrix

Example of virtual function:
// Step 10: Compute element sti
ness/force (ke, foe (fre: source term; fNe: Neumann BC))
virtual void Calculate_ElementStiffness_Force() = 0;

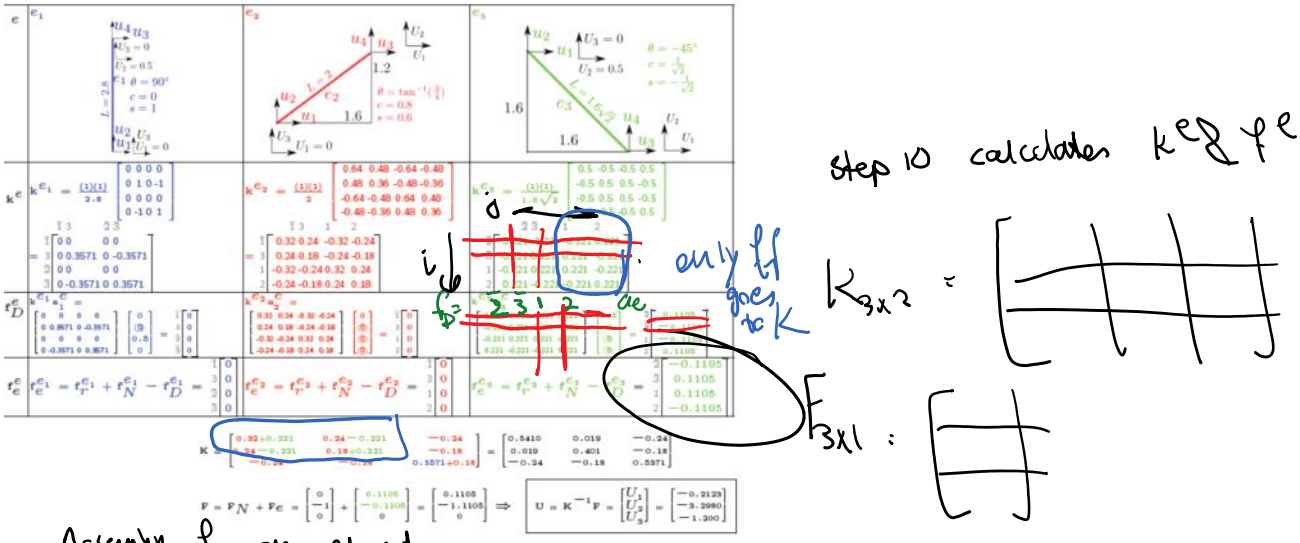
```
class PhyElementBar : public PhyElement
{
public:
    virtual void setGeometry();
    virtual void setInternalMaterialProperties(PhyMaterial*
    pMat);
    virtual void Calculate_ElementStiffness_Force();
    virtual void SpecificOutput(ostream& out) const;
    double L;
    double A;
    double E;
};

void PhyElementBar::Calculate_ElementStiffness_Force()
{
    // compute stiffness matrix:
    ke.resize(2, 2);
    double factor = A * E / L;
    ke(0, 0) = ke(1, 1) = factor;
    ke(1, 0) = ke(0, 1) = -factor;
}
```

Step 11: Assembly from local to global system

All the elements have the same form of assembly to the global system

Truss example: Assembly of global system



for local index $i = 1 : nedof$

global index $I = dofMap(i)$

for 4 → $dofMap = [\bar{2}\ \bar{3}\ 1\ 2]$

if $I < 0$ // I prescribed

Continue;

end

~~$fI = fe(i)$~~

recall $f^e = f_0^e - f_D^e$

for $j = 1 : nedof$

$J = dofMap(j)$

if $J > 0$

$K(I,J) = K(I,J) + ke(i,j);$

else

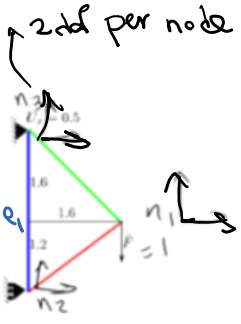
$fD(i) = fD(i) + ke(i,j) * qe(j);$ $\quad f_D^e = k_a^e q^e$

end

$fee(i) = fee(i) - fD(i);$

$F(I) = F(I) + fee(i);$

end



$nf = 3\ \ np = 3\ \ n = 6$

$ndf = 3 \times 2 = 6$

$np = 3 \rightarrow nf = ndof - np = 3$

node	P	pos	V	f
1	0	1		0
	0	2		−1
2	1	$\bar{1}$	0	0 ?
	0	3		0
3	1	$\bar{2}$	0.5	0 ?
	1	$\bar{3}$	0.6	0 ?

step 6

U_p (nf×1) = [1, 2, 3]

F_p (np×1) = [$\bar{1}$, $\bar{2}$, $\bar{3}$]

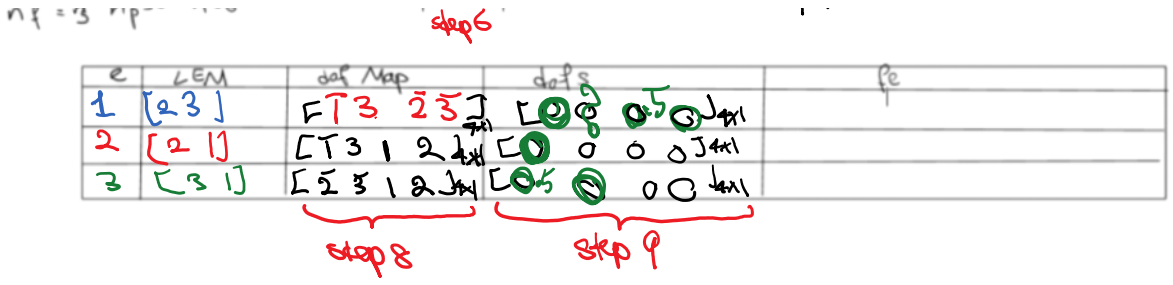
$K_{nf \times nf}$ (size nf)

↓ updated resp 1st

$f = F_n$ (np×1) = [0, −1, 0]

step 7

e	LEM	dof Map	dofs	fe



Step 11: Assembly from local to global system

- $\mathbf{K}$ and $\mathbf{F}$ (global stiffness and force) are already sized and set to zero.
- Element level (local) stiffness and force is calculated (Step 10).
- Element local to global dof $\mathbf{M}_t^e$ is already set (Step 8).
- Using dof map, we assemble local values to global values.
- Clearly, only free dofs are added to stiffness matrix and force vector.
- Element dof values (dofs: $\mathbf{a}^e$ is also set (Step 9).
- $\mathbf{f}_D^e = \mathbf{k}^e\mathbf{a}^e$ may not need to be formed and can be directly added to $\mathbf{f}_e^e$.

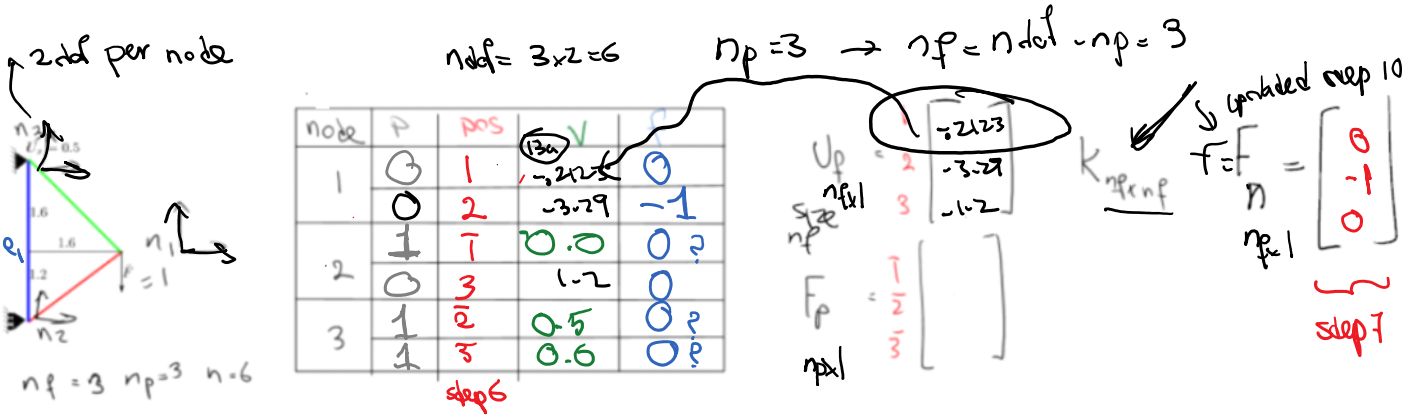
```
for e = 1:ne loop over elements
  fee = feo element total force = element all forces except essential force
  for i = 1:nedof loop over rows of ke; nedof = element # dof
    I = dofMap(i) local to global dof map M_t^e
    if (I > 0) I corresponds to a free dof, we skip prescribed dofs
      for j = 1:nedof loop over columns of ke
        J = dofMap(j) global dof corresponding to j
        if (J > 0) now both I and J are free and can add ke(i,j) to global K
          K(I, J) = K(I, J) + ke(i, j)
        else J < 0, prescribed dof j; add contributions of f_D^e = k^e a^e to f_e^e
          fee(i) = fee(i) - ke(i, j) * edofs(j) edofs: element dofs = a^e
        end
      end
      F(I) = F(I) + fee(i)element's total force fee component i'th is computed→added to F(I)
    end
  end
end
```


Step 12: Solve global (free) dof a from $\mathbf{Ka} = \mathbf{F}$

- Two major computational costs during FEM solve are:
  1. Assembly: Refers to: all node, element, and dof set up; computation of local ke and fee; assembly of those to global system. This step scales linearly versus n_e (ne)
  2. Linear algebra solution: $\mathbf{Ka} = \mathbf{F}$: We solve for unknown $\mathbf{a}$. Although conceptually simple, this step is a major source of computational cost. It scales higher than linear versus n_e $\Rightarrow$ As the problem size increases this term becomes more dominant.
- Solution of $\mathbf{Ka} = \mathbf{F}$:
  - WE DO NOT OBTAIN a from $\mathbf{a} = \mathbf{K}^{-1}\mathbf{F}$; We do not invert $\mathbf{K}$.
  - We only solve the problem for the specific RHS of $\mathbf{F}$.
  - In Comparison $\mathbf{K}^{-1}$ corresponds to the solution of $\mathbf{Ka} = \mathbf{F}$ for n_f RHS of $\mathbf{F} = e_i$, $i = 1, \ldots, n_f$ where n_f is the number of rows (and columns) of $\mathbf{K}$.
  - We employ methods such as LU factorization that computationally only solve the problem for the given RHS $\mathbf{F}$.
  - We take advantage of the structure of stiffness matrix: symmetry, bandedness, sparsity in choosing the right solution technique.
  - order of free dofs affects band of the matrix → various algorithms reorder free dofs such that the matrix band get smaller and the solution cost is optimized.
  - In your term projects you can simply employ simply compute

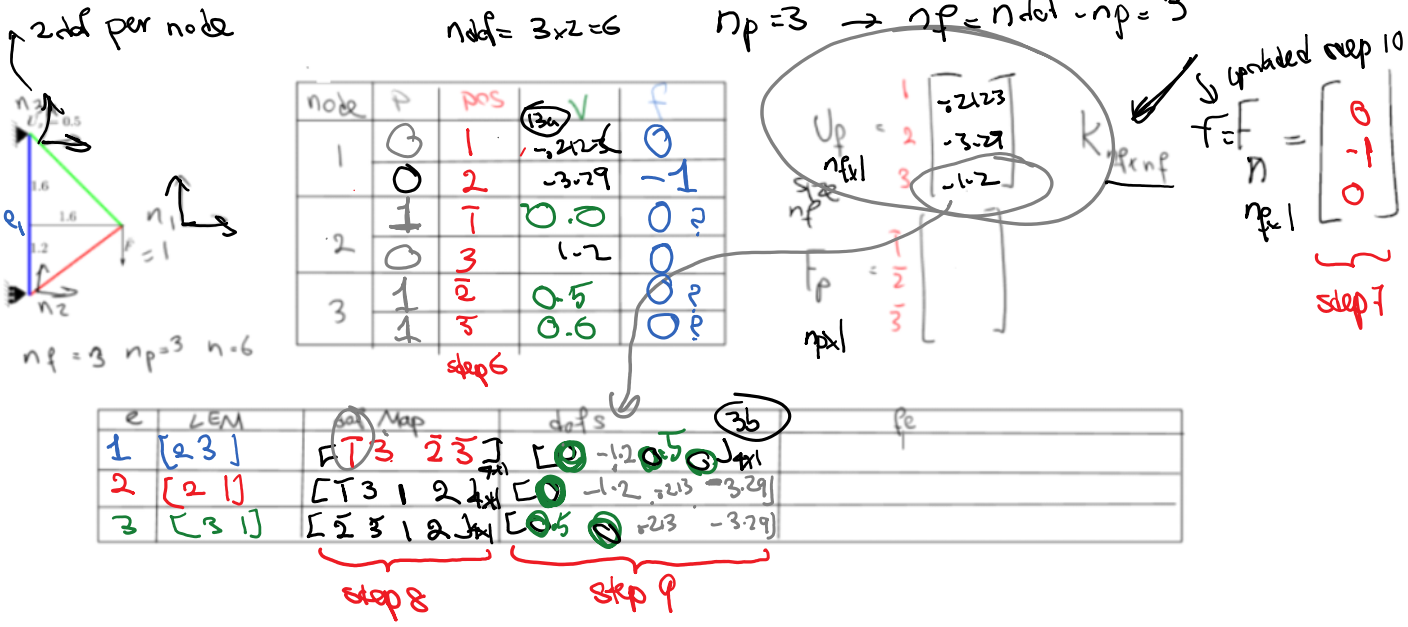
$a = K\backslash F$;

Step 13: Assign a to nodes and elements




```
for n = 1:nNodes
  for dofi = 1: node(n).nndof num dof for node (n)
    if node(n).ndof(dofi).p == false free dof
      posn = node(n).ndof(dofi).pos position of dof in global free F
      node(n).ndof(dofi).v = dofs(posn) set free dof val to corresponding val in global dofs (a)
    end
  end
end
```

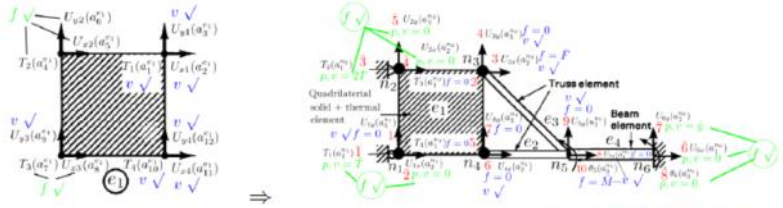
435 / 456



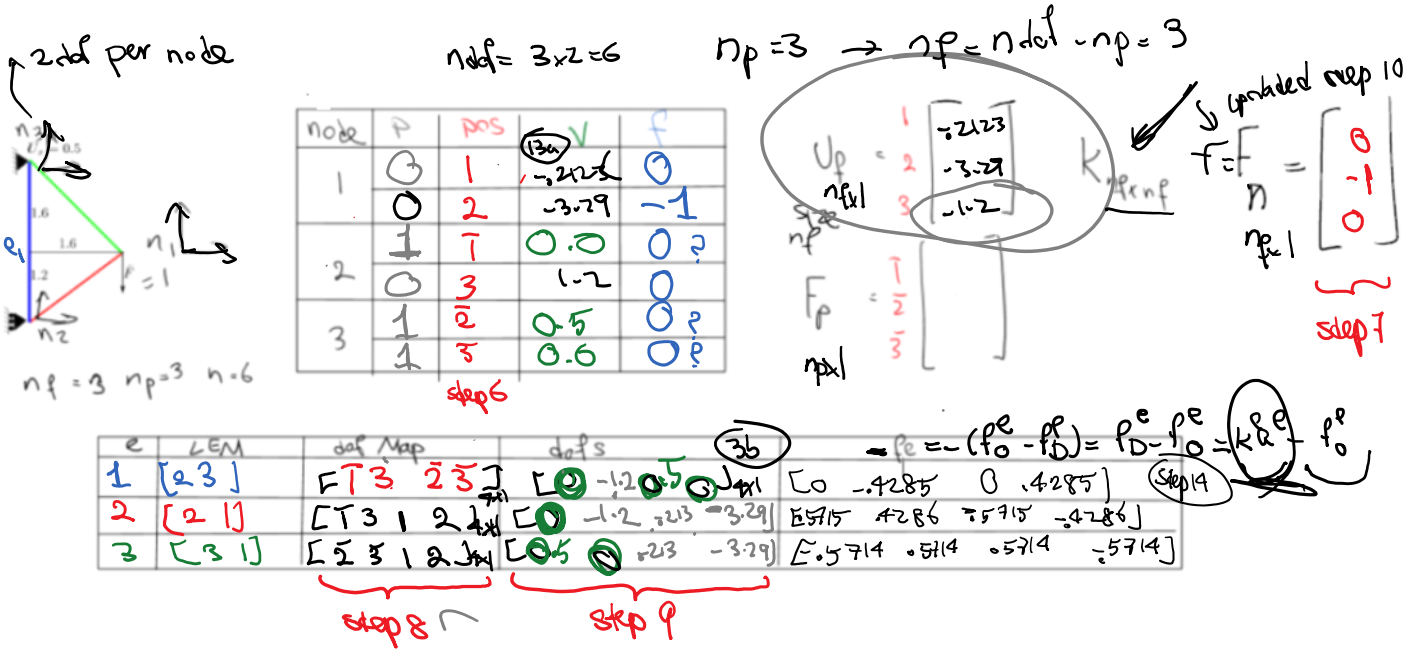
```
for e = 1:ne loop over elements
  for i = element(e).nedof loop over element dofs; nedof = # dof (n^e_dof)
    posn = element(e).dofMap(i) corresponding global position using dofMat (M^e_t)
    if (posn > 0) free dof
      element(e).edofs(i) = dofs(posn)
        set free element dof a^e to corresponding val in global dofs (a)
    end
  end
end
```

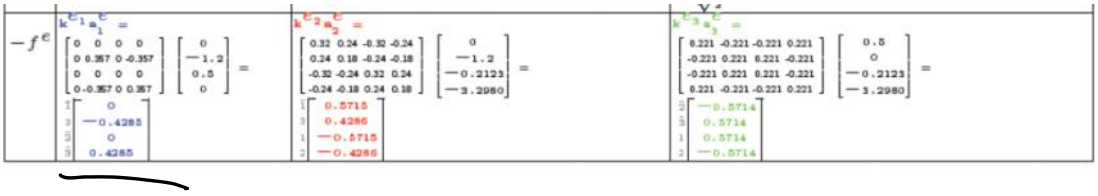

Remaining things: Element forces and prescribed dof forces in global structure

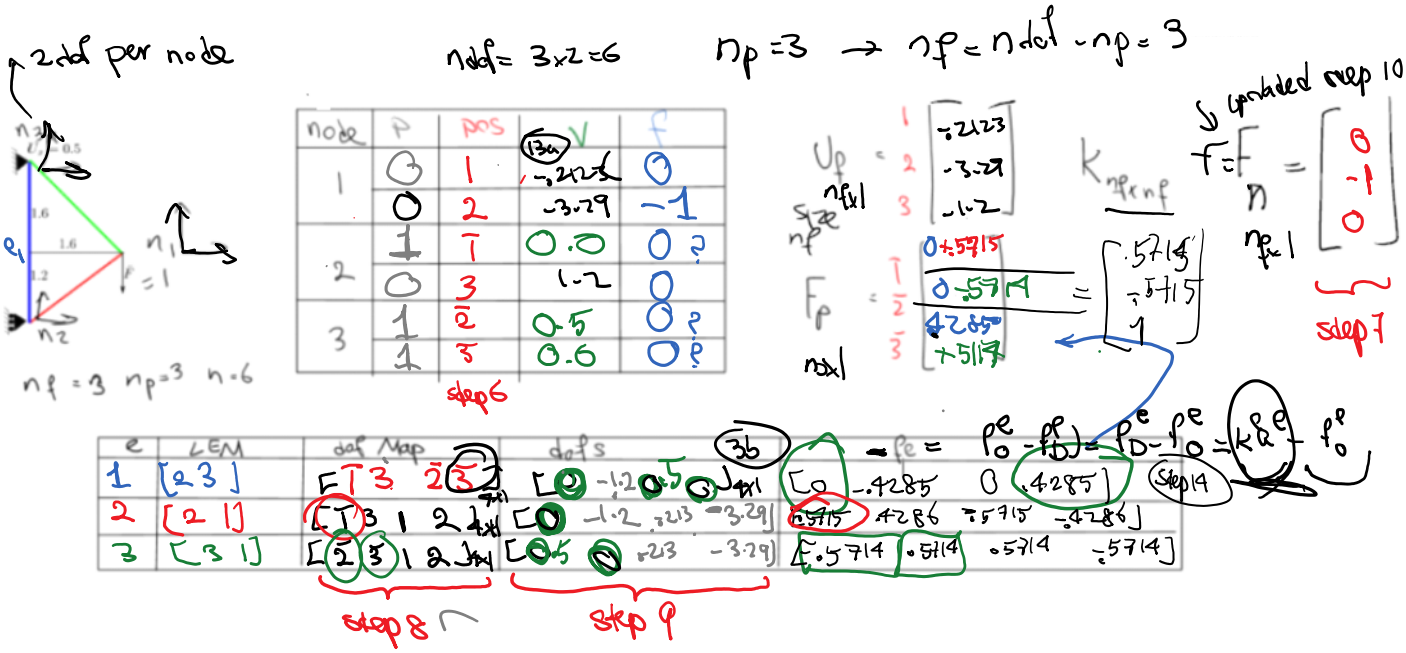
Step 14: Compute prescribed dof forces



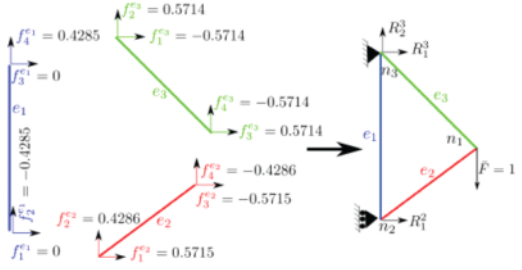
- At global nodes, the only unknowns at this stage are prescribed forces (reaction forces for solid mechanics).
- To set global nodal prescribed forces we need to:
  1. **Compute all element forces**: $\mathbf{f}^e_o$ (all forces but $\mathbf{f}^e_D$ is calculated), we add this to contributions from nodal dof values (previously known prescribed and newly solved free dofs).
  2. **Add element nodal forces to get global nodal forces**: The contribution (sum) of all element forces at a given prescribed dof is equal to the prescribed force (reaction force) at that dof. For convenient we store these values in $\mathbf{F}_p$ (n_p vector).
  3. **Assign prescribed global nodal dof forces** from their corresponding values in $\mathbf{F}_p$.
- **loop over elements → steps 1 & 2; loop over nodes → step 3**.







Truss Example: Reaction Forces



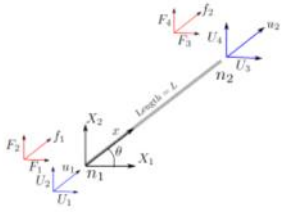
- First, we compute reaction forces by adding up forces from individual elements that contribute to reaction forces:

$$R_1^2 = f_1^{e1} + f_1^{e2} = 0 + 0.5715 = 0.5715 \tag{397a}$$

$$R_1^3 = f_3^{e1} + f_1^{e3} = 0 + -0.5714 = -0.5714 \tag{397b}$$

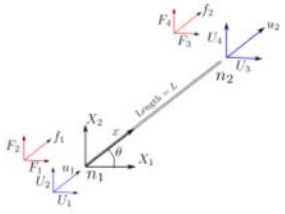
$$R_2^3 = f_4^{e1} + f_2^{e3} = 0.4285 + 0.5714 = 0.9999 \tag{397c}$$

Step 15: Compute/output nodes & elements: b) elements



- The computation and output of an element is again a black box function.

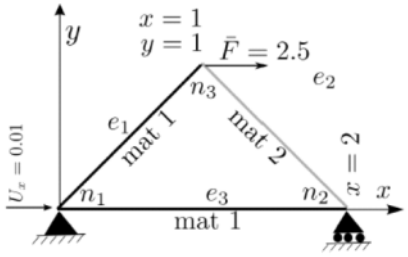
Step 15: Compute/output nodes & elements: b) elements



- The computation and output of an element is again a black box function.

Input file format

```
dim 2
ndofpn 2
Nodes
nNodes 3
id crd
1 0 0
2 2 0
3 1 1
Elements
ne 3
id elementType matID neNodes eNodes
1 3 1 2 1 3
2 3 2 2 3 2
3 3 1 2 1 2
PrescribedDOF
np 3
node node_dof_index value
1 1 0.01
1 2 0
2 2 0
FreeDOFs
nNonZeroForceFDOFs 1
node node_dof_index value
3 1 2.5
Materials
nMat 2
id numPara Paras
1 2 100 1
2 2 200 2
```



$E_1 = 100, A_1 = 1$
$E_2 = 200, A_2 = 2$

Output file format

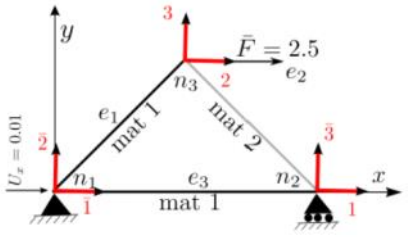
- Main FEMSolver function would be:

  function FEMSolver(runName, verboseOutput)

- runName is the name of the run (*e.g.*, TrussA).
- FEMSolver opens a file with name runName.txt to read the problem (with the format conforming to the sample on slide 455).
- This function, computes and outputs results in file runName.out.
- verboseOutput: if it is true more data will be output (explained on the next slide).
- Format of output file is
  1. Nodes: Node dof values and forces (plus dof position and prescribed boolean with verbose option).
  2. Elements: All element nodal forces (plus element specific output; *e.g.*, axial force for bars and trusses, *etc.*).

Output file format

```
Nodes
nNodes 3
id crd
values
forces
position(verbose)
prescribed_boolean(verbose)
1 0 0
a1_1 a1_2
F1_1 F1_2
-1 -2 (verbose)
1 1 (verbose)
2 2 0
a2_1 a2_2
F2_1 F2_2
1 -3 (verbose)
0 1 (verbose)
a3_1 a3_2
F3_1 F3_2
2 3 (verbose)
0 0 (verbose)
Elements
ne 3
id elementType
forces(verbose)
specific output
1 3
fee1_1 fee1_2 fee1_3 fee1_4 (verbose)
Te1
2 3
fee2_1 fee2_2 fee2_3 fee2_4 (verbose)
Te2
3 3
fee3_1 fee3_2 fee3_3 fee3_4 (verbose)
Te3
```



- lines with (verbose) are only output for verboseOutput == 1. Obviously (verbose) is not printed in either case and is only printed for clarity here.
- ai_j: is value (solution) for node i dof number j;*e.g.*, a3_1 is x displacement at node 3 ($x = 1, y = 1$).
- Fi_j: is force for node i dof number j;*e.g.*, F3_1 is x force at node 3 (which should be equal to 2.5, why?)
- feei_j: is total force (foe + fDe) for element i dof number j;*e.g.*, fee3_1 is the x force at its left node (global n_1).
- Last item of element output is specific to its type.
- For 2 node bar and truss elements Tei is the axial force in the element.

For the truss example in the class
TrussTestOutput.txt

```
Nodes
nNodes   3
id    crd
values
forces
position(verbose)
prescribed_boolean(verbose)

1     1.6   1.2
-0.212127       -3.29812
0     -1
0     1
0     0
```

2 0 0
0 -1.2
0.571429 0
0 2
1 0

3 0 2.8
0.5 0
-0.571429 1
1 2
1 1

Elements
ne 3
id ElementType
forces(verbose)
specific output
1 3
-0 -0.428571 -0 0.428571
0.428571

2 3
0.571429 0.428571 -0.571429 -0.428571
-0.714286

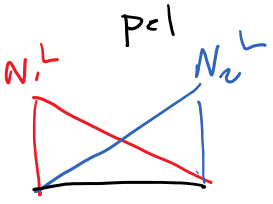
3 3
-0.571429 0.571429 0.571429 -0.571429
0.808122

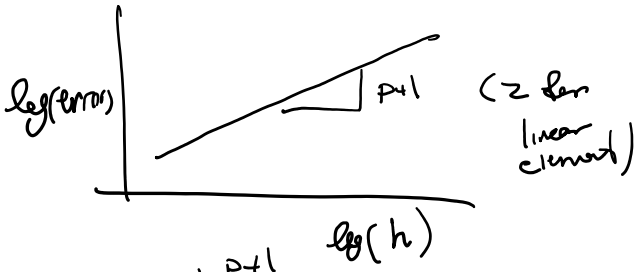
global_unknowns-dofs
size 3 -0.212127 -3.29812 -1.2
Support_forces-Fp
size 3 0.571429 -0.571429 1
K
rows3 cols 3
0.540971 0.0190291 -0.24
0.0190291 0.400971 -0.18
-0.24 -0.18 0.537143

F
size 3 0.110485 -1.11049 0

Higher order 1D, 2D, 3D elements

Higher order 1D elements

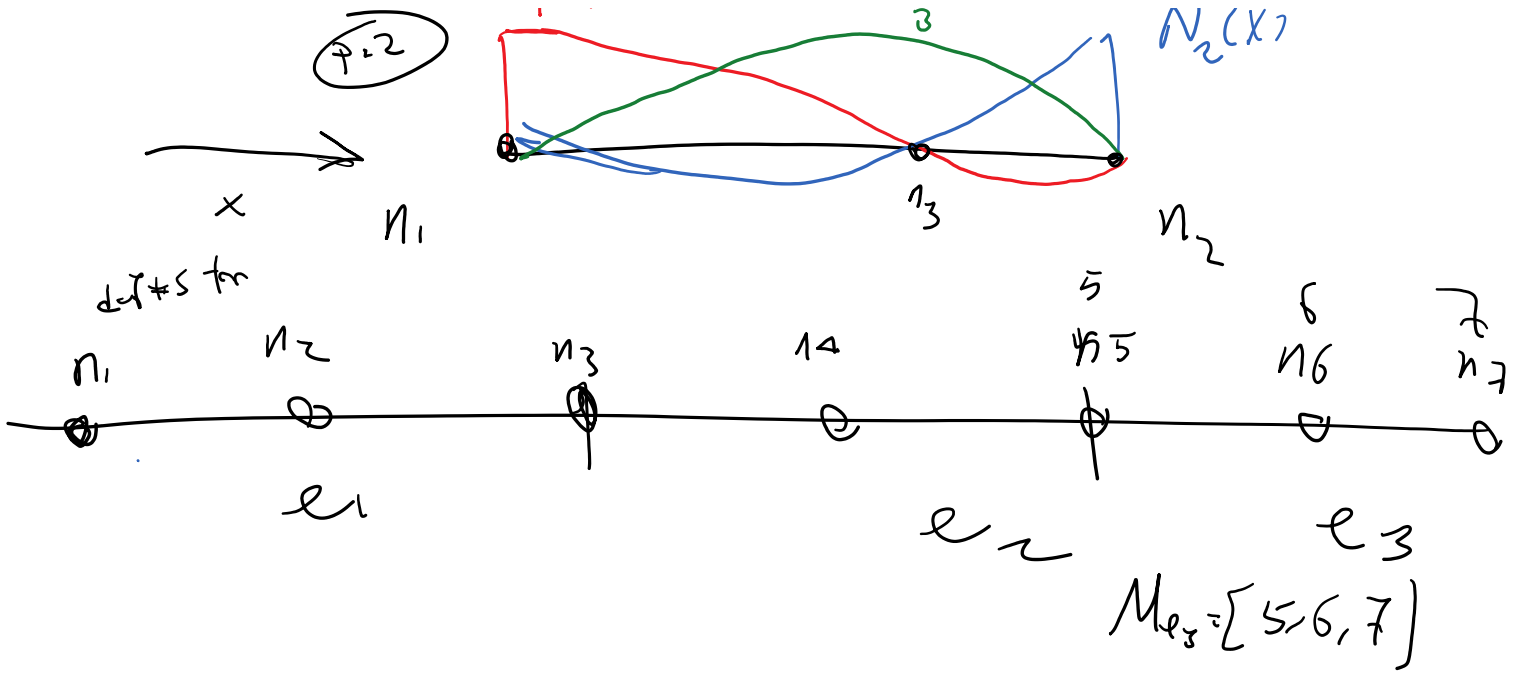




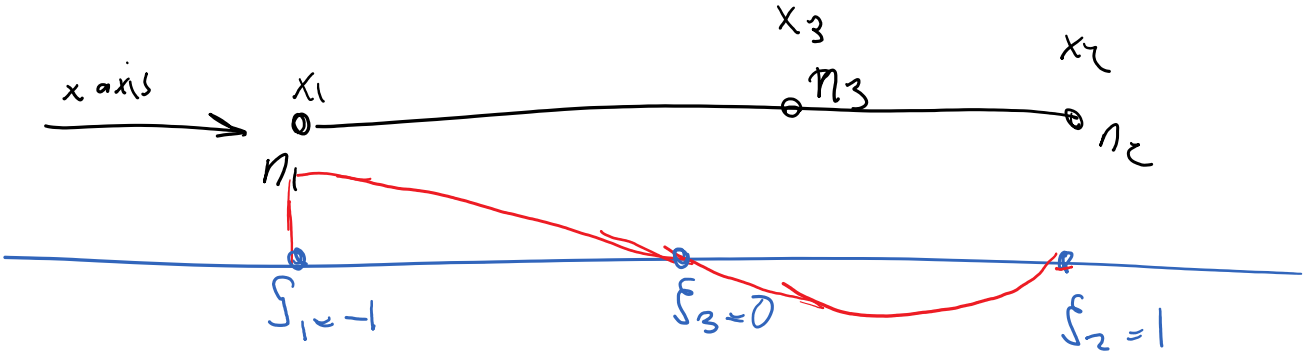
$error = Ch^{p+1}$

1D p=2 element

bubble shape function



Parent Geometry



$$N_1(\xi) = \frac{(\xi - \xi_2)(\xi - \xi_3)}{(\xi_1 - \xi_2)(\xi_1 - \xi_3)} \quad (L_1(\xi))$$

Lagrange polynomials

$N_1(\xi_1) = \frac{(\xi_1 - \xi_2)(\xi_1 - \xi_3)}{\text{denom}} = 1$ $\quad N_1(\xi_2) = \frac{(\xi_2 - \xi_2)(\xi_2 - \xi_3)}{\text{denom}} = 0$

$N_1(\xi_3) = 0$