2022/11/08

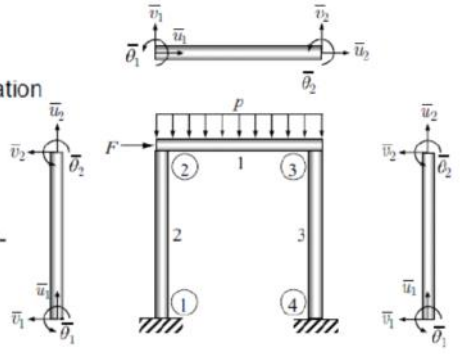
Tuesday, November 8, 2022 11:22 AM

Frames

Frames: 2D frame elements

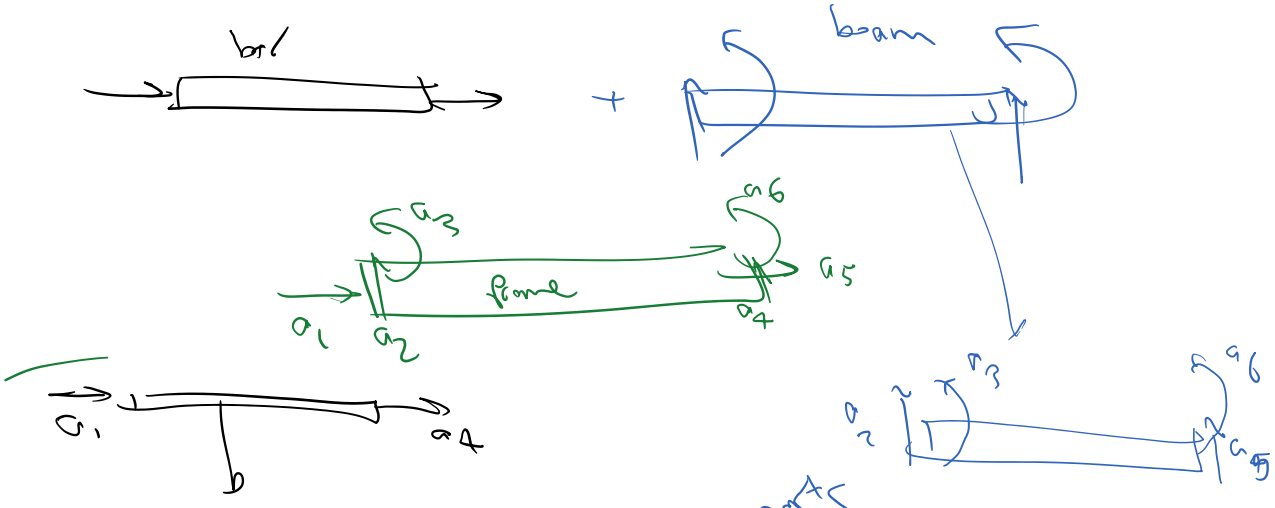
- Beam
  - Vertical deflection and slope. No axial deformation
- Frame structure
  - Can carry axial force, transverse shear force, and bending moment (Beam + Truss)
- Assumption
  - Axial and bending effects are uncoupled
  - Reasonable when deformation is small
- 3 DOFs per node

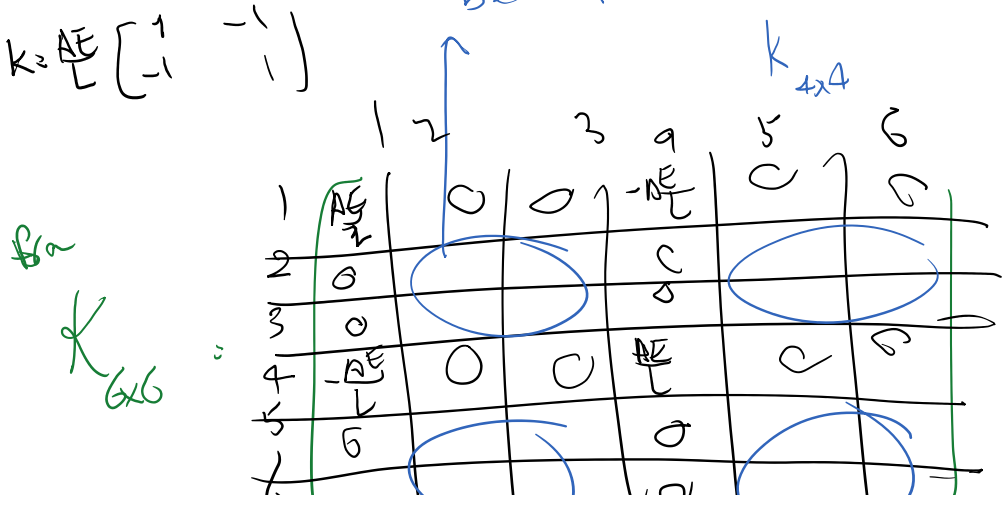
  $\{u_i, v_i, \theta_i\}$
- Need coordinate transformation like plane truss

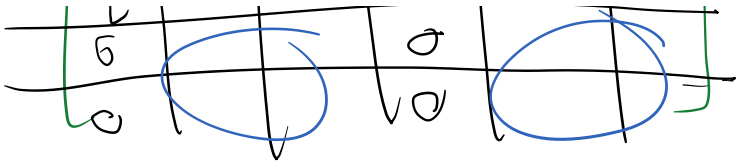


source: Nam-Ho Kim, Raphael T. Haftka; http://www2.mae.ufl.edu/nkim/eml5526/ sec. 5

381 / 456







2D frame element: axial and bending stiffness matrices

- Axial deformation (in local coord.)

$$\frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} \overline{u}_1 \\ \overline{u}_2 \end{Bmatrix} = \begin{Bmatrix} f_{\overline{x}1} \\ f_{\overline{x}2} \end{Bmatrix}$$

- Beam bending

$$\frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} \overline{v}_1 \\ \overline{\theta}_1 \\ \overline{v}_2 \\ \overline{\theta}_2 \end{Bmatrix} = \begin{Bmatrix} f_{\overline{y}1} \\ \overline{c}_1 \\ f_{\overline{y}2} \\ \overline{c}_2 \end{Bmatrix}$$

- Basically, it is equivalent to overlapping a beam with a bar
- A frame element has 6 DOFs

source: Nam-Ho Kim, Raphael T. Haftka; http://www2.mae.ufl.edu/nkim/eml5526/ sec. 5

383 / 456

2D frame: local & global coordinate stiffness matrices

- Element matrix equation (local coord.)

$$\begin{bmatrix} a_1 & 0 & 0 & -a_1 & 0 & 0 \\ 0 & 12a_2 & 6La_2 & 0 & -12a_2 & 6La_2 \\ 0 & 6La_2 & 4L^2a_2 & 0 & -6La_2 & 2L^2a_2 \\ -a_1 & 0 & 0 & a_1 & 0 & 0 \\ 0 & -12a_2 & -6La_2 & 0 & 12a_2 & -6La_2 \\ 0 & 6La_2 & 2L^2a_2 & 0 & -6La_2 & 4L^2a_2 \end{bmatrix}\begin{Bmatrix} \overline{u}_1 \\ \overline{v}_1 \\ \overline{\theta}_1 \\ \overline{u}_2 \\ \overline{v}_2 \\ \overline{\theta}_2 \end{Bmatrix} = \begin{Bmatrix} \overline{f}_{x1} \\ \overline{f}_{y1} \\ \overline{c}_1 \\ \overline{f}_{x2} \\ \overline{f}_{y2} \\ \overline{c}_2 \end{Bmatrix} \qquad a_1 = \frac{EA}{L} \qquad a_2 = \frac{EI}{L^3}$$

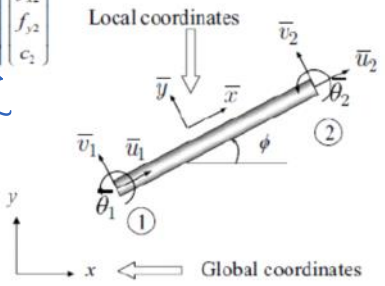
2D frame element: coordinate transformation

- Element-fixed local coordinates $\bar{x}-\bar{y}$
- Local DOFs $\{\bar{u},\bar{v},\bar{\theta}\}$ Local forces $\{\bar{f}_{\bar{x}},\bar{f}_{\bar{y}},\bar{c}\}$
- Transformation between local and global coord.

$$\begin{Bmatrix} f_{\bar{x}1} \\ f_{\bar{y}1} \\ \bar{c}_1 \\ f_{\bar{x}2} \\ f_{\bar{y}2} \\ \bar{c}_2 \end{Bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\phi & \sin\phi & 0 \\ 0 & 0 & 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} f_{x1} \\ f_{y1} \\ c_1 \\ f_{x2} \\ f_{y2} \\ c_2 \end{Bmatrix}$$

$$\{\bar{\mathbf{f}}\} = [\mathbf{T}]\{\mathbf{f}\}$$

$$\{\bar{\mathbf{q}}\} = [\mathbf{T}]\{\mathbf{q}\}$$



source: Nam-Ho Kim, Raphael T. Haftka; http://www2.mae.ufl.edu/nkim/eml5526/ sec. 5

382 / 456

2D frame: local & global coordinate stiffness matrices

- Element matrix equation (local coord.)

$$\begin{bmatrix} a_1 & 0 & 0 & -a_1 & 0 & 0 \\ 0 & 12a_2 & 6La_2 & 0 & -12a_2 & 6La_2 \\ 0 & 6La_2 & 4L^2a_2 & 0 & -6La_2 & 2L^2a_2 \\ -a_1 & 0 & 0 & a_1 & 0 & 0 \\ 0 & -12a_2 & -6La_2 & 0 & 12a_2 & -6La_2 \\ 0 & 6La_2 & 2L^2a_2 & 0 & -6La_2 & 4L^2a_2 \end{bmatrix} \begin{Bmatrix} \bar{u}_1 \\ \bar{v}_1 \\ \bar{\theta}_1 \\ \bar{u}_2 \\ \bar{v}_2 \\ \bar{\theta}_2 \end{Bmatrix} = \begin{Bmatrix} \bar{f}_{x1} \\ \bar{f}_{y1} \\ \bar{c}_1 \\ \bar{f}_{x2} \\ \bar{f}_{y2} \\ \bar{c}_2 \end{Bmatrix}$$

$$a_1 = \frac{EA}{L} \qquad a_2 = \frac{EI}{L^3}$$

$$[\bar{\mathbf{k}}]\{\bar{\mathbf{q}}\} = \{\bar{\mathbf{f}}\}$$

- Element matrix equation (global coord.)

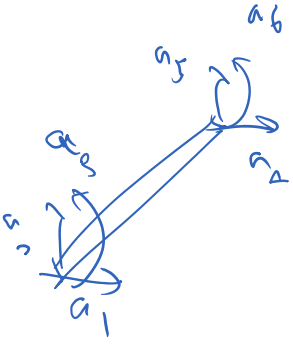
$$[\bar{\mathbf{k}}][\mathbf{T}]\{\mathbf{q}\} = [\mathbf{T}]\{\mathbf{f}\} \Longrightarrow [\mathbf{T}]^T[\bar{\mathbf{k}}][\mathbf{T}]\{\mathbf{q}\} = \{\mathbf{f}\} \Longrightarrow [\mathbf{k}]\{\mathbf{q}\} = \{\mathbf{f}\}$$

$$[\mathbf{k}] = [\mathbf{T}]^T[\bar{\mathbf{k}}][\mathbf{T}]$$

- Same procedure for assembly and applying BC

source: Nam-Ho Kim, Raphael T. Haftka; http://www2.mae.ufl.edu/nkim/eml5526/ sec. 5

384 / 456



HW

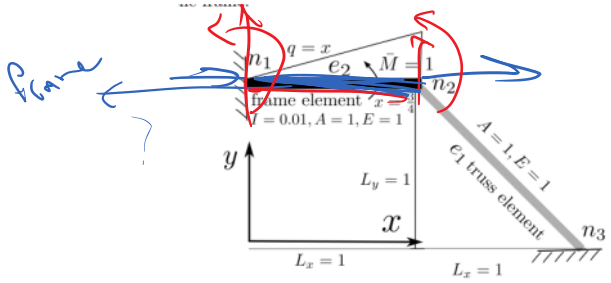


Figure 3: Frame and truss example.

but treat it as if there a bar & beam element at the same locali

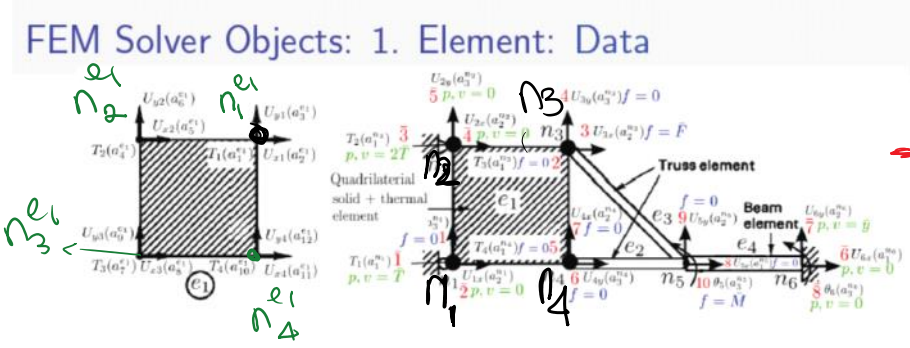
---

Coding:

For your term project you can use the incomplete C++ code I provide (10% extra credit) or use Matlab, Python (5%), etc. to write your code.
Need to read and write in the formats I provide
You can work in groups of 2 or 3 (C++, Python)

Part A:
Classes relevant to FEM (Object-oriented programming)

For each class, we discuss Data and functions



= dof numbers

Data

- id
- nNodes
- eNodes(LEM)

e1
1
4
[3 , 2 , 1, 4]

- edof (a) : vector of unknowns [T1, Ux1, Uy1 | T2, Ux2 Uy2 | T3 Ux3 Uy3 | T4 Ux4 Uy4]

- dofMap (M) : [2 3 4 | 3 4 5 | 1 2 1 | 5 6 7]

- dofMap (M) : [2 3 4 | 3 7 5 | 1 2 1 | 5 6 7]

- forces $f_e^e, f_0^e, f_D^e, f_N^e, \ldots$

$f_e^e = (f_\sigma^e + f_N^e + \ldots) - f_D^e$

f_0^e → other

f_D^e → Dirichlet

for term project $f_0^e = 0$

- Stiffness k^e

- eType Square

- {Physics} → list of physics the element solves
  Solid, Thermal

Examples of functions

① Virtual Calculate Stiffness

Different functions for different element types

bar $k^e = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ truss $k^e = \frac{AE}{L}\left[\begin{array}{c|c} k_b & -k_b \\ \hline -k_b & k_b \end{array}\right]_{4\times4}$

$K_{beam} = \frac{AE}{L^3}\{\ \}_{4\times4}$

② Assemble Stiffness

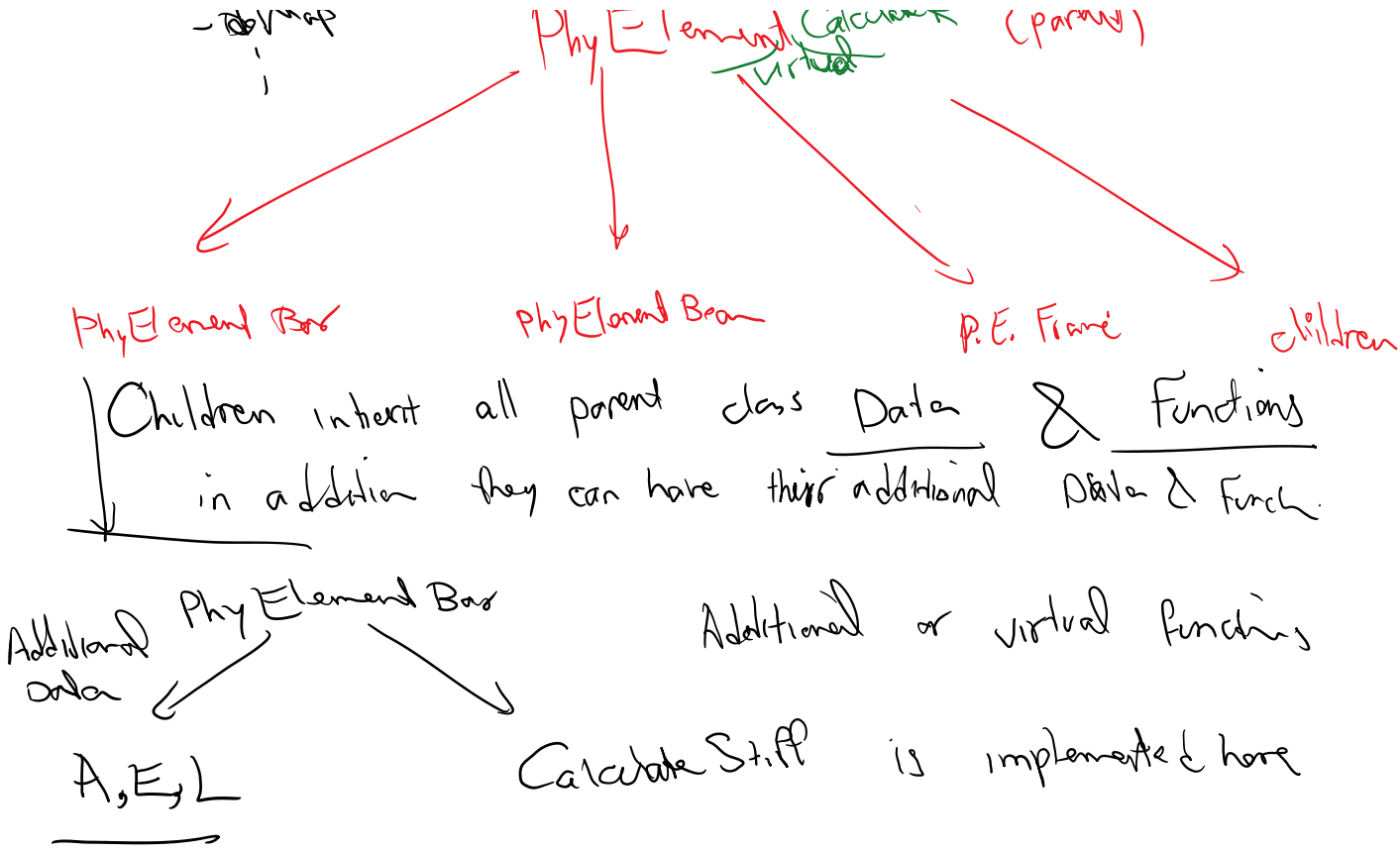
Same exact function for all element types

All needed dofMap → global K

Functions

Data
- id
- nNode
- dofMap

PhyElement → Assemble K → Calculate k (parallel) Virtual




```
class PhyElement
{
    friend ostream& operator<<(ostream& out, const PhyElement& dat);
public:
    virtual void setGeometry() = 0;
    virtual void setInternalMaterialProperties(PhyMaterial* pMat) = 0;
    void setNodeConnectivity_Sizes(int nNodeInElement, int ndofpnIn, vector<int>& eNodesIn,
    vector <PhyNode*>& eNodePtrsIn);

    void print(ostream& out);
    // Step 8: Element dof maps Me
    // Step 9: Set element dofs ae
    void setElementDofMap_ae(int ndofpn);

    // Step 10: Compute element sti
    ness/force (ke, foe (fre: source term; fNe: Neumann BC))
    virtual void Calculate_ElementStiffness_Force() = 0;
    // Step 11: Assembly from local to global system
    void AssembleStiffnessForce(MATRIX& globalK, VECTOR& globalF);

    // Step 14: Compute prescribed dof forces
    void UpdateElementForces_GlobalFp(VECTOR& Fp);

    // Step 15: Compute/output element specific data
    virtual void SpecificOutput(ostream& out) const {THROW("does not have implementation");};
```

children will provide their implementation

All element types are assembled the same way, so the implementation is here

```
    // Step 15: Compute/output element specific data
    virtual void SpecificOutput(ostream& out) const {THROW("does not have implementation");};

    int id;
    int neNodes; // # element nodes
    vector<int> eNodes; // element node vector
    vector <PhyNode*> eNodePtrs;
    int nedof; // # element dof
    VECTOR edofs; // element dofs
    vector<int> dofMap;
    ElementType eType;
    int matID;
    MATRIX ke; // element stiffness matrix
    VECTOR foe; // element force vector from all sources other than essential BC
    VECTOR fde; // element essential BC force
    VECTOR fee; // all element forces
};
```

Funcly

Data

Look a child of PhyElement

```
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;
};
```

PhyElementBar is a child of PhyElement

Whenever we go to subclasses (children) we only need to add new + virtual functions and new data. All the other things are inherited.

Element has its own declaration (C++, .h)

Implementation (C++, .cpp)

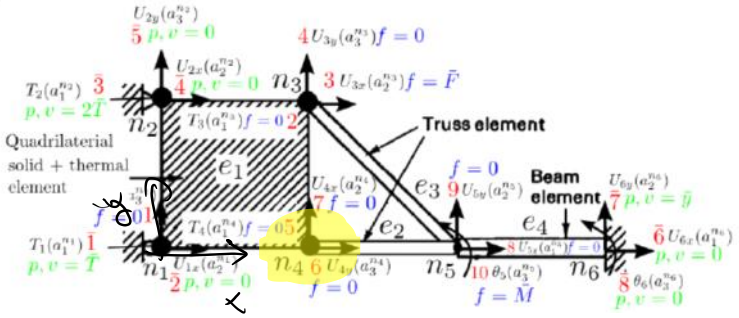
new data

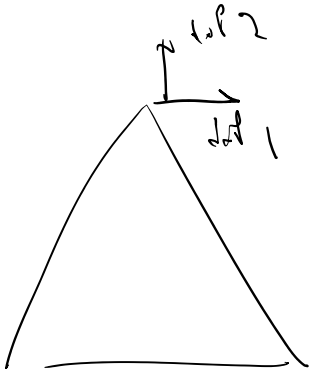
From PhyElement.cpp

```
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;
}
```

Other classes:

FEM Solver Objects: 4. Node: Data





Data

id

coordinate

dofs

nn dof

n 4

4

[1, 0]

{ PhyDofs }

3

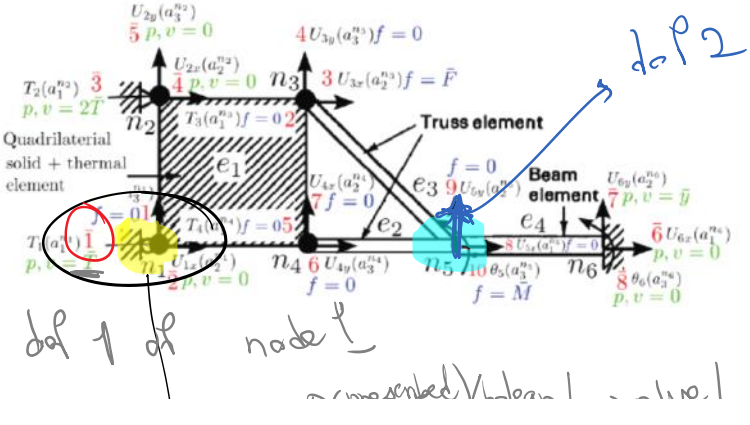
we have a vector of dofs

PhyNode.h

```
class PhyNode
{
public:
    void set_nndof(int nndofIn);
    void UpdateNodePrescribedDofForces(VECTOR& Fp);
    ID id;
    VECTOR coordinate;
    vector <PhyDof> ndof;
    int nndof;     // number of dofs
};
```

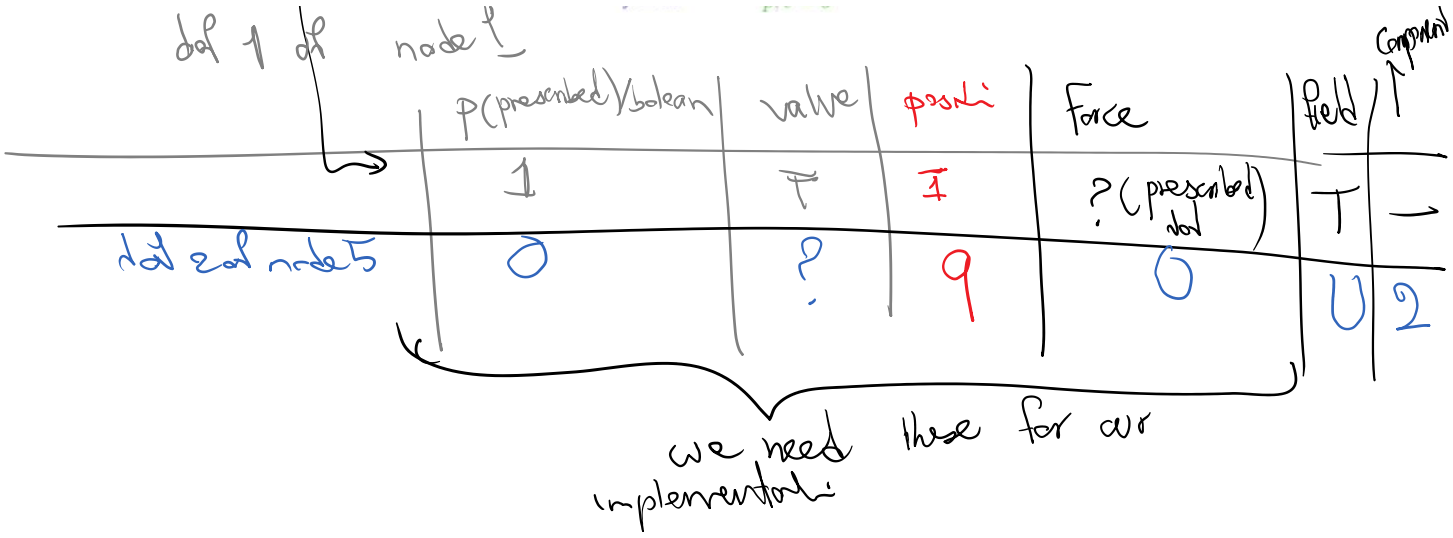
Data

FEM Solver Objects: 5. Dof: Data



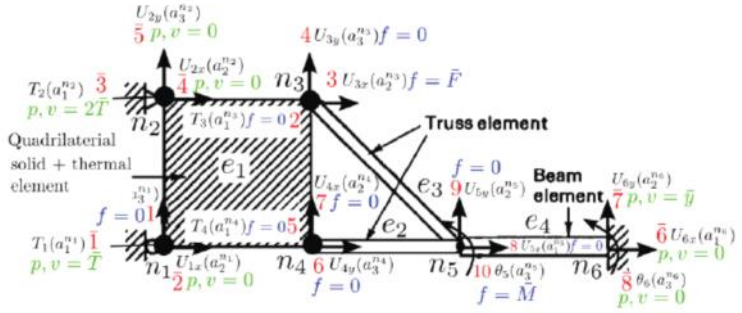
dof 2 of node 5

dof 1 of node 1



PhyDof.h

FEM Solver Objects: 5. Dof



Examples of dof for the structure shown are:

dof	p	pos	v	f	field	index
1 of n_1	true	$\bar{1}$	$\bar{T}$	unknown	T	-
3 of n_1	false	1	unknown	0	U	2
3 of n_5	false	10	unknown	$\bar{M}$	θ	- (a vector in 3D)
2 of n_6	true	$\bar{7}$	$\bar{y}$	unknown	U	2

399 / 456

class PhyDof
{
public:

```
    PhyDof();

    bool p;   // boolean: whether the dof is prescribed
    int pos;  // position in the global system (for free and
    prescribed)
    double v;      // value of dof
    double f;      // force corresponding to dof

//    F can be stress i can be (0, 1) sigma_{01}
//    Field      F;
//    INDEX i;
};
```

Steps for solution

Solution steps

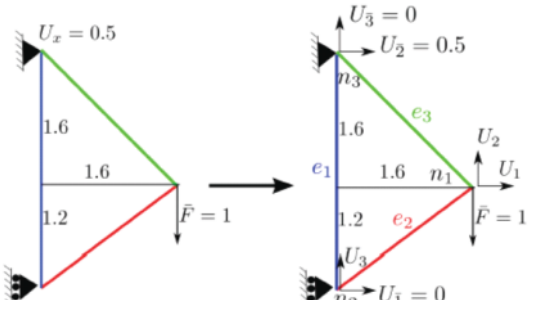
The steps for FEM solution are:

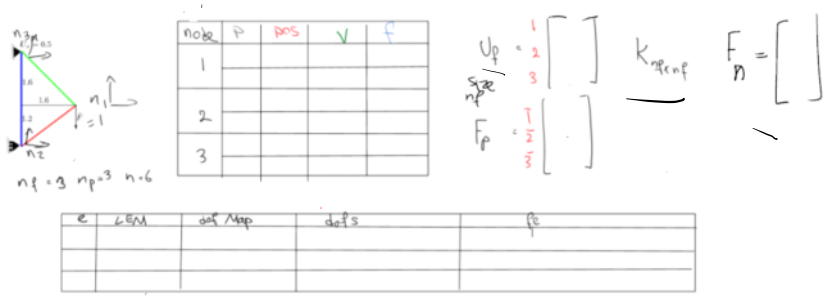
1. Set Element nodal dofs.
2. Set global dofs using element dofs.
3. Compute n_f from n_{dof} and n_p and resize and zero stiffness matrix and force vector.
4. Set global prescribed dofs.
5. Set global free dofs.
6. Set dof (free + prescribed) positions.
7. Set $\mathbf{F}(\mathbf{F}_f)$.
8. Set element dof maps $\mathbf{M}_t^e$.
9. Set element (prescribed) dofs.
10. Compute element stiffness matrix and force vectors.
11. Assemble element stiffnesses and forces to global system.
12. Solve for (free) dofs $\mathbf{a}$ from $\mathbf{Ka} = \mathbf{F}$.
13. Assign $\mathbf{a}$ to nodes and elements.
14. Compute prescribed dof forces: $\mathbf{F}_p$ (if needed).
15. Compute (if needed) output nodes and elements.

402 / 456

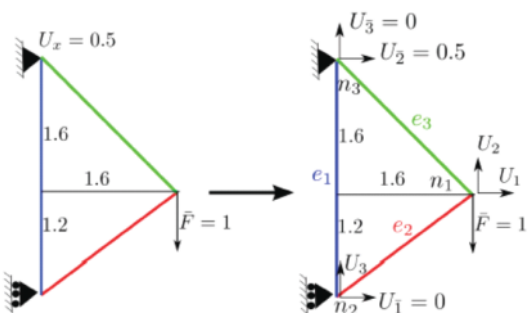
Input text file
TrussTest.txt
It's for this problem

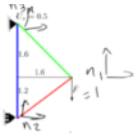
Truss Example





Truss Example

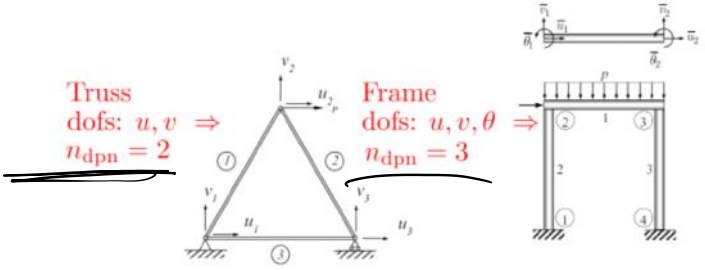




1				
2				
3				

e	LEM	dof Map	dofs	fe

Steps 1 & 2: Simplified limited case



- FEM implementation become considerably simpler for problems where all elements are of the same type (regardless of number of physics per element).
- In this case, we define:

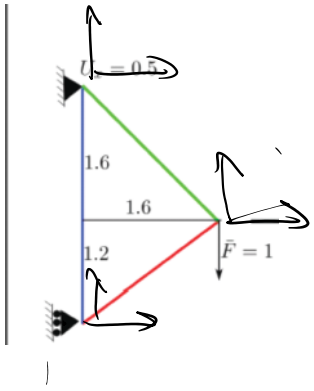
$$n_{\text{dpn}} := \text{Number of dof. per node denoted by ndofpn} \quad (448)$$

- There would be identity map between element nodal dof and global nodal dofs. That is, there is the same set of dofs used for both.
- Figure above shows two of such examples:

410 / 456

TrussTest.txt

dim 2
ndofpn 2



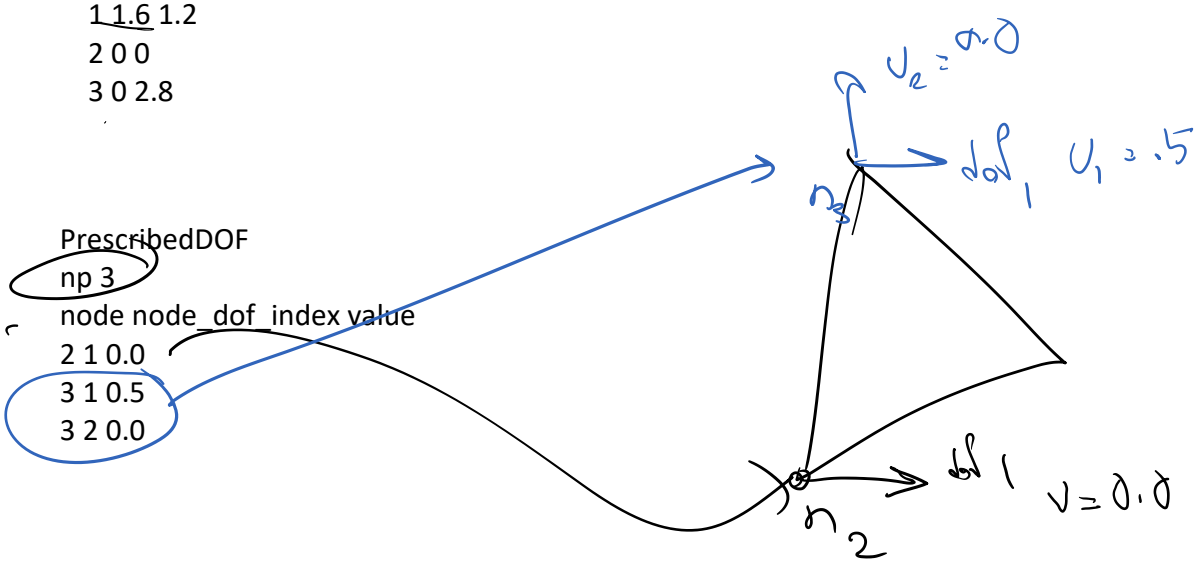
Step 3: Set global number of dofs, stiffness, and force.

$n = nnod * ndofpn$

$3 * 2 = 6$

Nodes
nNodes 3
id crd
1 1.6 1.2
2 0 0
3 0 2.8

PrescribedDOF
np 3
node node_dof_index value
2 1 0.0
3 1 0.5
3 2 0.0



$n_p = 3$

$n = 6$

$n_f = n - n_p = 6 - 3 = 3$

K_{ff} $_{n_f \times n_f}$ $=$ [ ]

(F_p) $_{n_p \times 1}$ $=$ [ ]

$a_{n_p} =$ [ ]