FEM20241118

Monday, November 18, 2024 12:48 PM

From last time,

Functions of the Element class

CalculateElementStiffness

bar $k^e = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

truss $k^e = \frac{AE}{L}\left[\begin{array}{c|c} k^b & -k^b \\ \hline -k^b & k^b \end{array}\right]$

procedural

if (eType == bar)

{

$k = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

}

else if ( )

}

}

Parent (Element class)

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

Parent does not have the implementation

class PhyElementBar : public PhyElement

{

```
virtual void Calculate_ElementStiffness_Force();
```

```
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;
}
```

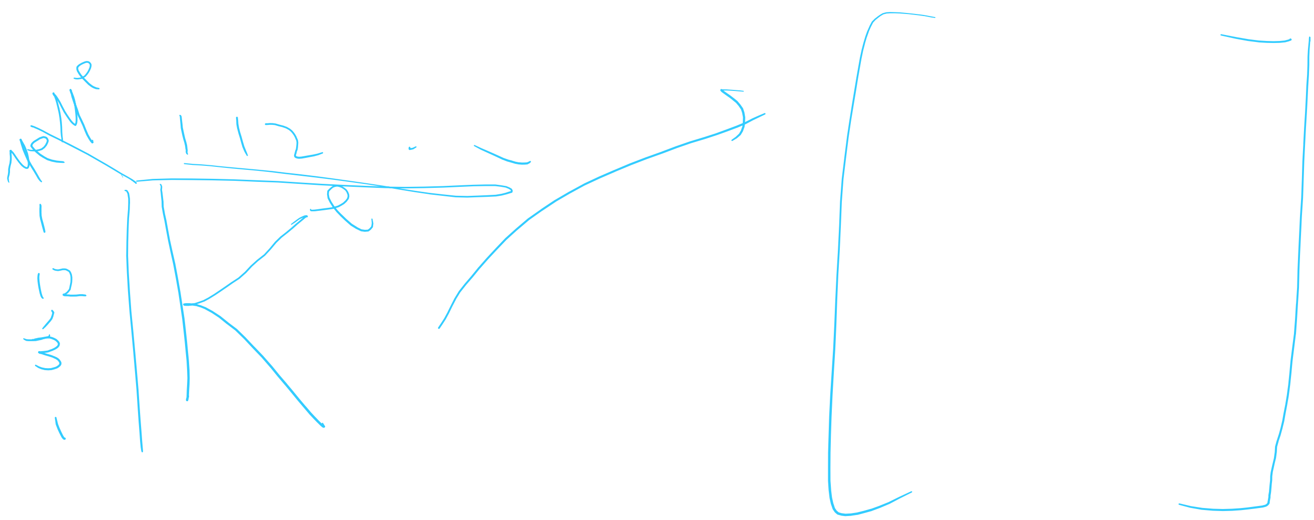
$$\frac{AE}{L}\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}$$

```
void PhyElementTruss::Calculate_ElementStiffness_Force()
{
    // compute stiffness matrix:
    ke.resize(4, 4);
    double factor = A * E / L;
    for (int I = 0; I < 2; ++I)
        for (int J = 0; J < 2; ++J)
        {
            double f2 = factor;
            if ((I + J) % 2 != 0)
                f2 = -factor;
            ke(2 * I, 2 * J) = c * c * f2;
            ke(2 * I + 1, 2 * J) = ke(2 * I, 2 * J + 1) = c * s * f2;
            ke(2 * I + 1, 2 * J + 1) = s * s * f2;
        }
    cout << "ke\n" << ke << endl;
}
```

In contrast to this virtual function (each subclass can have its own implementation) we have normal functions.

K global K

Example:
Assembly



Independent of the element -> can be implemented at the element level.

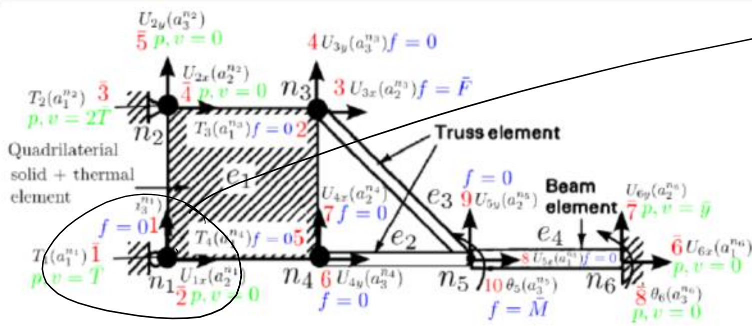
At the element element, we have an implementation:

```
// Step 11: Assembly from local to global system
void AssembleStiffnessForce(MATRIX& globalK, VECTOR& globalF);
```

```cpp
void PhyElement::AssembleStiffnessForce(MATRIX& globalK, VECTOR& globalF)
{
    fee.resize(nedof);
    if (foe.size() == nedof)
        fee = foe;
    else
        fee = 0.0;

    int I, J;
    for (int i = 0; i < nedof; ++i)
    {
        I = dofMap[i];
        if (I < 0) // prescribed dof
            continue;
        for (int j = 0; j < nedof; ++j)
        {
            J = dofMap[j];
```

FEM Solver Objects: 4. Node: Data



- id: Clearly, id of n_1 is 1.
- coordinate: *e.g.*, for n_1 the coordinate in the figure can be:

$$\mathrm{crd}_{XY}(n_1) = [0 \quad 0]$$

coordinate components are alway represented with respect to a coordinate system (another geometry object we will not further discuss herein).

- {ndof}: *i.e.*, a "set" of dofs. Dof is a class being described next. It includes data such as being free or prescribed, position in global free or prescribed dofs, value (*e.g.*, displacement), and force.
- nndof: Number of dof for the given node.

We will not discuss functions for the node object.

397 / 456

id 1

coord [0, 0]

nndof = 3

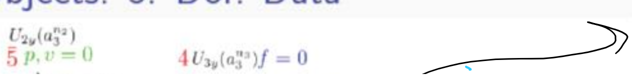
ndof =

{ Temperature → U_x U_y }

```cpp
class PhyNode
{
    // no need to have friend specification, because all members are public
    // in fact, we can even declare the function outside the class scope (because we don't need to give fri
    // remember friendship is only given inside the class
//  friend ostream& operator<<(ostream& out, PhyNode& node);
//  friend ostream& operator<<(ostream& out, PhyNode& node);

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

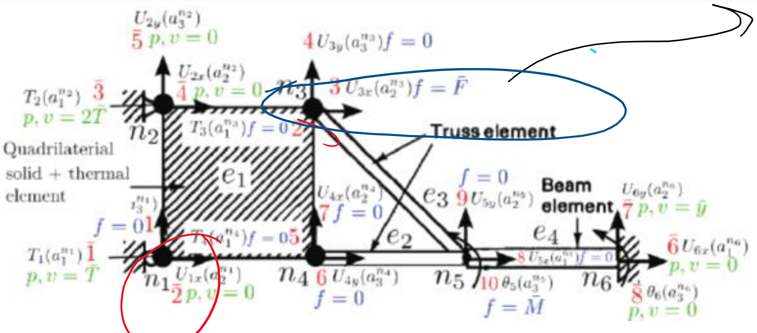
FEM Solver Objects: 5. Dof: Data



field U

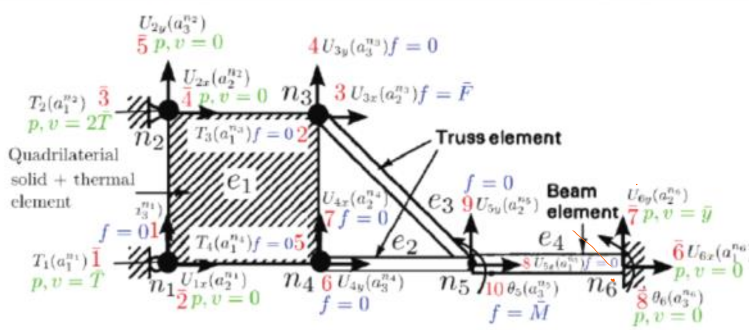
element (x)

FEM Solver Objects: 5. Dof: Data



field U
component 1 (x)
prescribed false
pos 3
Value (actual U_x)
force $\bar{F}$

FEM Solver Objects: 5. Dof



Examples of dof for the structure shown are:

dof	p	pos	v (value)	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

free dof


```
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;
};
```

Solution steps

The steps for FEM solution are:

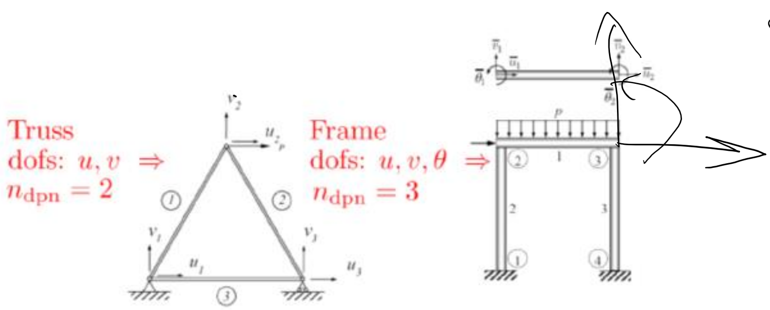
1. Set Element nodal dofs.
2. Set global dofs using element dofs.
3. Compute n_f from n_dof and n_p and resize and zero stiffness matrix and force vector.
4. Set global prescribed dofs.
5. Set global free dofs.
6. Set dof (free + prescribed) positions.
7. Set $\mathbf{F}(\mathbf{F}_f)$.
8. Set element 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 a from $\mathbf{K}\mathrm{a} = \mathbf{F}$.
13. Assign 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

We only code a simple case:

bar

nddpn = 1



- 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:

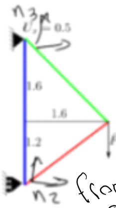
$$\boxed{n_\mathrm{dpn} := \text{Number of dof. per node denoted by ndofpn}} \qquad (448)$$

- 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.



$f = 1$ ndofpn
$\times$ nNodes
from file

$n_f = 3 \quad n_p = 3 \quad n = 6$

$n_f = n - n_f = 3$

node	p	pos	v	f
1				
2				
3				

unknown vector

(U_f) size n_f = 1, 2, 3 $\begin{bmatrix} \\ \\ \end{bmatrix}$

(F_p) n_p = 1, 2, 3 $\begin{bmatrix} \\ \\ \end{bmatrix}$

$K_{n_f \times n_f}$

$F_n = \begin{bmatrix} \\ \\ \end{bmatrix}$

e	LEM	dof Map	dofs	fe

nnode = 3

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

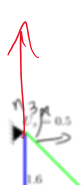
n = ndofpn $\times$ nNodes = 6

By default every dof is free with zero force ->
Provide prescribed dofs

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

Provide free dofs with nonzero forces

Step 4: Set global prescribed nodal dof



node	p	pos	v	f
1	false		0	0
	false		0	0

unknown vector

(U_f) size = 1, 2, 3 $\begin{bmatrix} \\ \\ \end{bmatrix}$

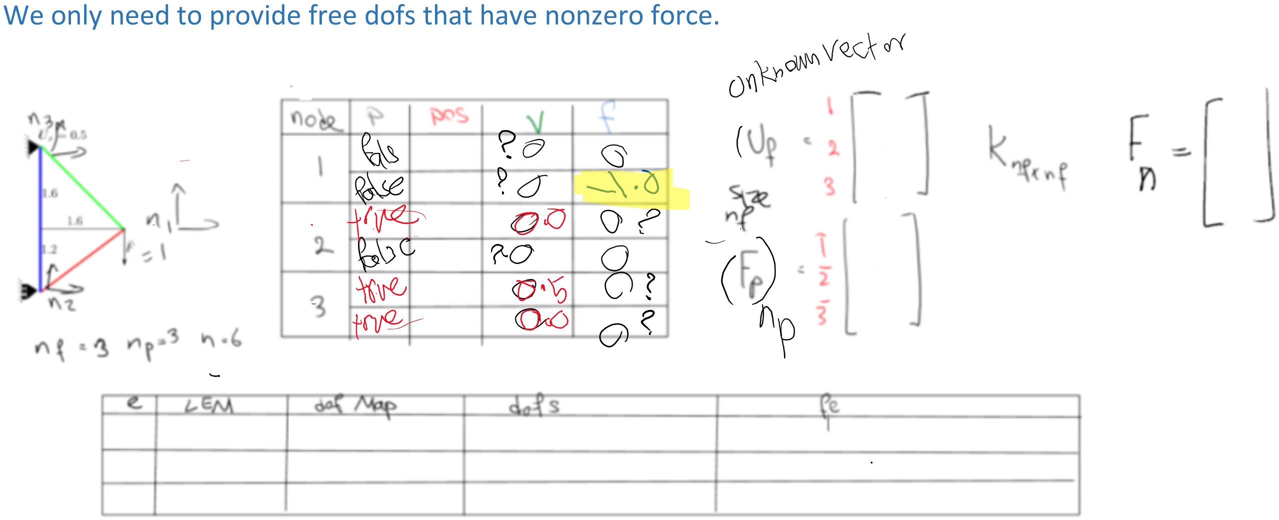
$K_{n_f \times n_f}$

$F_n = \begin{bmatrix} \\ \end{bmatrix}$

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

Step 5: Set global free nodal dof

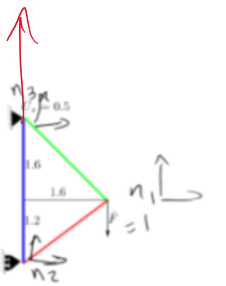
We only need to provide free dofs that have nonzero force.



FreeDofs
nNonZeroForceFDOFs 1
node node_dof_index value
1 2 -1.0

Step 6:

Step 6: dof positions



$n_f = 3 \quad n_p = 3 \quad n = 6$

node	p	pos	v	f
1	false	1	? 0	0
	false	2	? 0	-1.0
2	true	1	0.0	0 ?
	false	3	? 0	0
3	true	2	0.5	0 ?
	true	3	0.0	0 ?

unknown vector

$(U_f) = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$ size n_f $\qquad K_{n_f \times n_f} \qquad F_n = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

$(F_p)_{n_p} = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

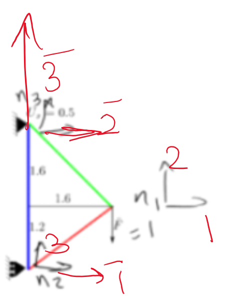
e	LEM	dof Map	dofs	fe

freeCntr = ~~0~~ → 1 → 2 → 3

prescribedCntr = ~~0~~ → 1 → 2 →

Step 7: Set $\mathbf{F}(\mathbf{F}_n)$

or F_p



$n_f = 3 \quad n_p = 3 \quad n = 6$

node	p	pos	v	f
1	false	1	? 0	0
	false	2	? 0	-1.0
2	true	1	0.0	0 ?
	false	3	? 0	0
3	true	2	0.5	0 ?
	true	3	0.0	0 ?

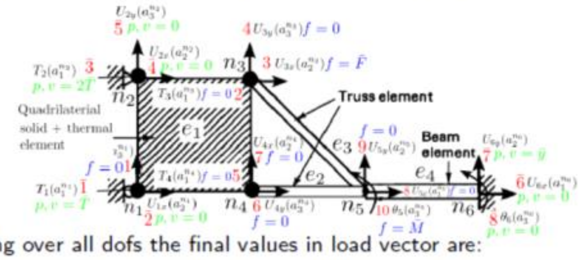
unknown vector

$(U_f) = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$ size n_f $\qquad K_{n_f \times n_f} \qquad F_n = \begin{bmatrix} 0 \\ -1.0 \\ 0 \end{bmatrix}$

$(F_p)_{n_p} = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

e	LEM	dof Map	dofs	fe

Step 6: dof positions; Step 7: Set $\mathbf{F}(\mathbf{F}_f)$



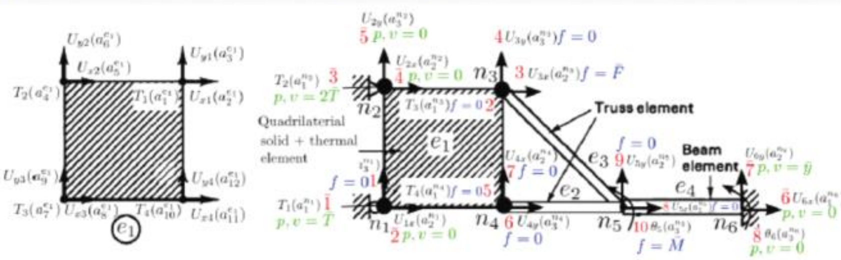
- After looping over all dofs the final values in load vector are:

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & \bar{F} & 0 & 0 & 0 & 0 & 0 & 0 & \bar{M} \end{bmatrix}$$

```
posf = 0, posp = 0
for n = 1:nNodes
  for dofi = 1: node(n).nndof num dof for node (n)
    if node(n).ndof(dofi).p == true prescribed dof
      posp = posp - 1;
      node(n).ndof(dofi).pos = posp;
    else free dof
      posf = posf + 1;
      node(n).ndof(dofi).pos = posf;
      F(posf) = node(n).ndof(dofi).f
    end
  end
end
```

$\longrightarrow$ step 7 $(F_n \quad F_e)$

Step 8: Element dof maps $\mathbf{M}_t^e$



- As mentioned, $\mathbf{M}_t^e$ is a vector of size n^e_{dof} that maps elements dofs to global positions.
- For element 1, dofs are ordered as (loop over nodes, then loop over dofs for the node):

$$\begin{aligned} a_1^e &= \begin{bmatrix} a_1^{e1} & a_2^{e1} & \cdots & a_{12}^{e1} \end{bmatrix} \\ &= \begin{bmatrix} T_1 & U_{x1} & U_{y1} \mid T_2 & U_{x2} & U_{y2} \mid T_3 & U_{x3} & U_{y3} \mid T_4 & U_{x4} & U_{y4} \end{bmatrix} \end{aligned}$$

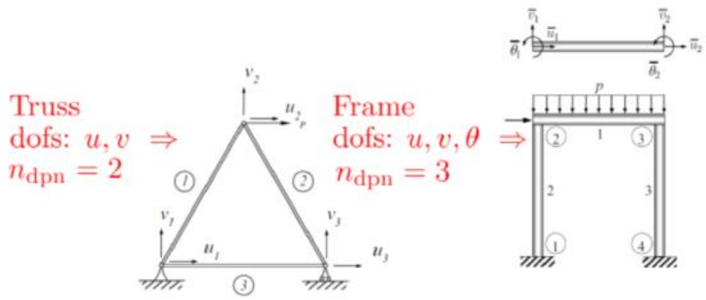
- We need to map these dofs to global dofs and have their position in $\mathbf{M}_t^e$ vector. For example, 1st dof of node 1 ($a_1^{e1} = T_1$) is mapped to first dof of n_3 which has position 2.
- 2nd dof of node 3 ($a_8^{e1} = U_{y2}$) is mapped to 2nd dof of n_1 which has position $\bar{2}(-2)$.
- The map for element e_1 is:

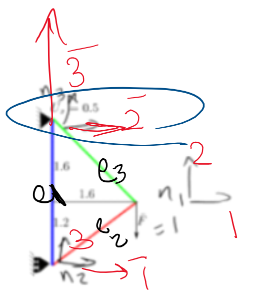
$$\mathbf{M}^{e_1}{}_t = \begin{bmatrix} 2 & 3 & 4 & \bar{3} & \bar{4} & \bar{5} & \bar{1} & \bar{2} & 1 & 5 & 6 & 7 \end{bmatrix}$$

Step 8: Element dof maps M_t^e: Simplified limited case

Truss dofs: u, v ⇒ $n_{dpn} = 2$

Frame dofs: u, v, θ ⇒ $n_{dpn} = 3$





$n_f = 3 \quad n_p = 3 \quad n = 6$

node	P	pos	V	f
1	false	1	? 0	0
	false	2	? 0	-1.0
2	true	-1	0.0	0 ?
	false	3	? 0	0
3	true	-2	0.5	0 ?
	true	-3	0.0	0 ?

unknown vector

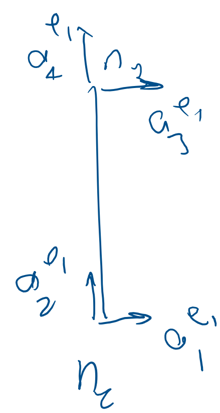
$(U_f) = \begin{matrix}1\\2\\3\end{matrix}\begin{bmatrix} \\ \\ \end{bmatrix}$ size n_f

$(F_p) = \begin{matrix}\bar{1}\\\bar{2}\\\bar{3}\end{matrix}\begin{bmatrix} \\ \\ \end{bmatrix}$ n_p

$K_{n_f \times n_f}$

$F_n = \begin{bmatrix} 0 \\ -1.0 \\ 0 \end{bmatrix}$

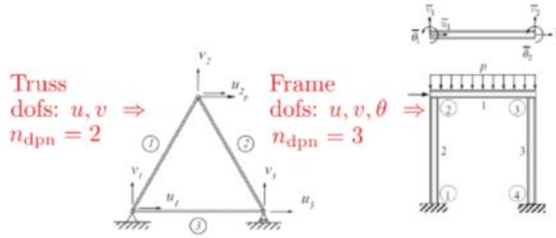
e	LEM (eNodes)	dof Map	dofs	fe
e1	[2,3]	[-1,3,-2,-3]		
	(2,1)	[-1,3,1,2]		
	(3,1)	[-2,-3,1,2]		



ne 3
id elementType matID neNodes eNodes
1 3 1 2 2 3
2 3 1 2 2 1
3 3 1 2 3 1

eType == 3 ⇒ element = truss

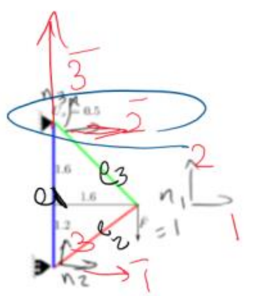
Step 8: Element dof maps $\mathbf{M}_t^e$: Simplified limited case



A simplified pseudo code looks like: **ecdof** $= 1$ dof counter for element

for en $= 1$: **neNodes** number of element nodes
  gn $= \text{LEM}(\textbf{en})$ global node number for element node en
  for endof $= 1$: **ndofpn** This number is fixed now, *e.g.*, 2 for 2D trusses
    dofMap(ecdof) = node(gn).dof(endof).pos
      gndof = endof, we bypass some steps here
    ecdof $=$ **ecdof** $+ 1$ increment counter
  end
end

Step 9: Set element dofs $\mathbf{a}^e$



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

node	P	pos	V	f
1	false	1	? 0	0
	false	2	? 0	-1.0
2	true	-1	0.0	0 ?
	false	3	? 0	0
3	true	-2	0.5	0 ?
	true	-3	0.0	0 ?

unknown vector

(U_f) size n_f $= \begin{matrix}1\\2\\3\end{matrix}\begin{bmatrix} \\ \\ \end{bmatrix}$

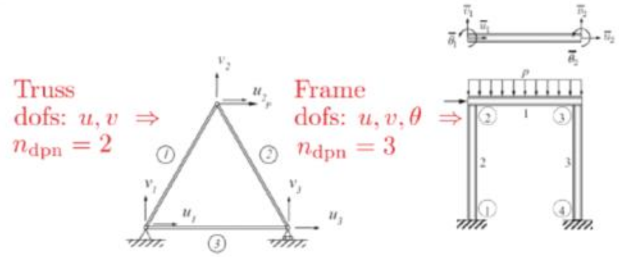
$(F_p)_{n_p} = \begin{matrix}\bar{1}\\\bar{2}\\\bar{3}\end{matrix}\begin{bmatrix} \\ \\ \end{bmatrix}$

$K_{n_f \times n_f}$

$F_n = \begin{bmatrix} 0 \\ -1.0 \\ 0 \end{bmatrix}$

e	LEM (nodes)	dof Map	dof ?	fe
e1	[2,3]	[-1,3,-2,-3]	[0.0, 0, 0.5, 0.0]	
	[2,1]	[-1,3,1,2]	[0, 0?, 0?, 0]	
	[3,1]	[-2,-3,1,2]	[0.5, 0, 0?, 0?]	

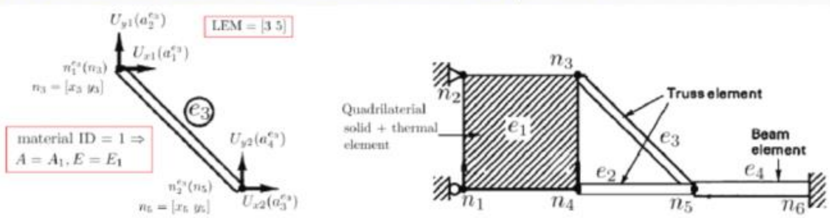
Step 9: Set element dofs $\mathbf{a}^e$: Simplified limited case



- Similar to steps 1, 2, and 8, step 9 can be greatly simplified if we assume all nodes share exactly the same set of dofs.
- Noting n_{dpn} (ndofpn) = Number of dof. per node, simplified merged steps 8 & 9 are:

```
dofs = zeros(nedof) element dofs (edof) resized to number of element dofs and zeroed
ecdof = 1 dof counter for element
for en = 1: neNodes number of element nodes
  gn = LEM(en) global node number for element node en
  for endof = 1: ndofpn This number is fixed now, e.g., 2 for 2D trusses
    if (node(gn).dof(endof).p == true)    gndof = endof, we bypass some steps here
      dofs(ecdof) = node(gn).dof(endof).value; e dof val = corresponding global val
    end
    dofMap(ecdof) = node(gn).dof(endof).pos
    ecdof = ecdof + 1    increment counter
  end
end
```


Step 10: Compute element stiffness/force



- Almost all assembly of element stiffness/forces are done in a "black box".
- That is, as far as assembly of local to global stiffnesses and forces are considered, there is no need to know how the local stiffness matrix is assembled.
- This is if an object oriented language is used, most of these functions are virtual (*i.e.*, different elements have different implementations for these computations under the same function name).
- For multiphysics problems the assembly of physics into the element is very much similar to assembly of elements into the global system: Each physics assembles its own part into element system which subsequently will be mapped to global system.
- Types of information needed before element stiffness/force computations:
  1. Element type: *e.g.*, truss or bar element; 1st, 2nd order, *etc.*.
  2. Geometry: Geometry of element (*e.g.*, L^e for trusses).
  3. Material properties: Properties needed such as A and E for trusses.


```
class PhyElementFrame : public PhyElement
{
public:
    virtual void setGeometry();
    virtual void setInternalMaterialProperties(PhyMaterial* pMat);
    virtual void Calculate_ElementStiffness_Force();
```

```cpp
void PhyElementFrame::Calculate_ElementStiffness_Force()
{
    a1 = E * A / L;
    a2 = E * I / pow(L, 3);
    // complete (student)
    //! 1. stiffness matrix in local coordinate system
    // complete kLocalCoordinate from 6 x 6 matrix from slide 384 / 457 of the course notes
    kLocalCoordinate.resize(6, 6);
    kLocalCoordinate = 0.0;
    kLocalCoordinate(0, 0) = a1, kLocalCoordinate(0, 3) = -a1, kLocalCoordinate(3, 0) = -a1, kLocalCoordinate(3, 3)
    double tmp = 12 * a2;
    kLocalCoordinate(1, 1) = tmp, kLocalCoordinate(1, 4) = -tmp, kLocalCoordinate(4, 1) = -tmp, kLocalCoordinate(4,
    tmp = 6 * L * a2;
```

```cpp
void PhyElementTruss::Calculate_ElementStiffness_Force()
{
    // compute stiffness matrix:
    ke.resize(4, 4);
    double factor = A * E / L;
    for (int I = 0; I < 2; ++I)
        for (int J = 0; J < 2; ++J)
        {
            double f2 = factor;
            if ((I + J) % 2 != 0)
                f2 = -factor;
            ke(2 * I, 2 * J) = c * c * f2;
            ke(2 * I + 1, 2 * J) = ke(2 * I, 2 * J + 1) = c * s * f2;
            ke(2 * I + 1, 2 * J + 1) = s * s * f2;
        }
    cout << "ke\n" << ke << endl;
}
```